leetro

MPC01 运动控制卡

操
作
手
册

（3.2 版）

乐创自动化技术有限公司
LEETRO AUTOMATION CO.,LTD.

目　　录

1　概　述.....1

1.1　MPC01 的软硬件简介.....1
1.2　MPC01 的结构.....2
1.3　MPC01 的技术特性和使用范围.....2
1.4　MPC01 的运动控制功能.....3
1.4.1　单轴运动控制.....3
1.4.2　多轴独立运动控制.....4
1.4.3　多轴插补运动控制.....4
1.4.4　中断、光码盘反馈和其它能力.....5
1.5　MPC01 型号说明.....5

2　控制卡的安装.....6

2.1　硬件安装.....6
2.1.1　设置控制卡基地址.....6
2.1.2　设置中断号.....7
2.1.3　设置减速开关的方向.....8
2.1.4　将控制卡插入PC机.....8
2.2　软件安装.....8

3　MPC01 接口.....10

3.1　开关量信号接口.....10
3.1.1　DB15 接口定义.....10
3.1.2　DB15 接线方法.....11
3.2　脉冲量信号接口.....12
3.2.1　DB25 接口定义.....12
3.2.2　DB25 接线方法.....13

4　运动控制系统的开发.....15

4.1　开发DOS环境下的运动控制系统.....15
4.1.1　链接运动函数库.....15
4.1.2　简单的定位运动例程.....15
4.1.3　简单的连续运动和回原点运动.....16
4.1.4　多轴插补运动举例.....18
4.1.5　中断产生及处理.....20
4.2　开发WINDOWS下的运动控制系统.....21
4.2.1　开发Visual Basic控制程序.....22
4.2.2　调用动态链接库函数.....22

4.2.3 动态链接库中的函数 22

5 函数描述 23

5.1 控制卡和轴设置函数 23
5.2 运动指令函数 29
5.2.1 独立运动函数 29
5.2.2 插补运动函数 31
5.3 制动函数 36
5.4 位置和状态查询函数 36
5.4.1 位置查询函数 36
5.4.2 状态查询函数 38
5.5 中断管理和其它函数 40

6 新增功能及函数 45

6.1 MPC01(3.2 版)软件概述 45
6.2 新增功能说明 45
6.2.1 运动指令执行方式 45
6.2.2 获取错误代码 47
6.3 运动控制系统的开发 47
6.3.1 MPC01（3.2 版）软件的使用要求 47
6.3.2 MPC01（3.2 版）软件的安装 47
6.3.3 开发Windows环境下的运动控制系统 47
6.4 扩展库函数描述 50
6.4.1 批处理状态设置函数 50
6.4.2 错误代码函数 51
6.4.3 其它新增函数 52
6.5 常见问题及解决方法 55
6.5.1 软件中对板卡进行初始化设置方法 55
6.5.2 读取编码器反馈 56
6.5.3 多指令连续运动 57
6.5.4 运动变速 57
6.5.5 正确判断前一个运动指令是否执行完毕 58
6.5.6 减速、原点信号的使用 60
6.5.7 如何提高速度精度 60
6.5.8 如何实现方向信号超前于脉冲信号 60
6.5.9 如何利用Z脉冲实现精确回原点 61
6.5.10 运行EXE文件时系统显示找不到DLL文件 61
6.5.11 如何将开发的软件系统制作成安装程序后发行给最终用户 61
6.5.12 运行EXE文件后死机 61
6.5.13 软件能够正常启动，但无法产生运动 61
6.5.14 如何在Windows 9X操作系统环境下正确设置MPC01 卡中断号 62

6.5.15 如何升级函数库 62
6.6 错误代码表 63
6.7 多指令连续运动时的升降速处理 64
6.7.1 功能说明 64
6.7.2 相应库函数描述 64
6.7.3 使用方法及应注意的问题 64
7 函数索引 66
8 附录 69
8.1 两轴步进控制系统示例 69
8.1.1 系统配置 69
8.1.2 控制电路接线图 69
8.2 单轴数字式伺服控制系统示例 70
8.2.1 系统配置 70
8.2.2 控制电路接线图 70
8.3 PC打印机口用作I/O口 72
8.4 PC机I/O地址分配 73
8.5 PC机中断线分配 74

1 概　述

1.1 MPC01 的软硬件简介

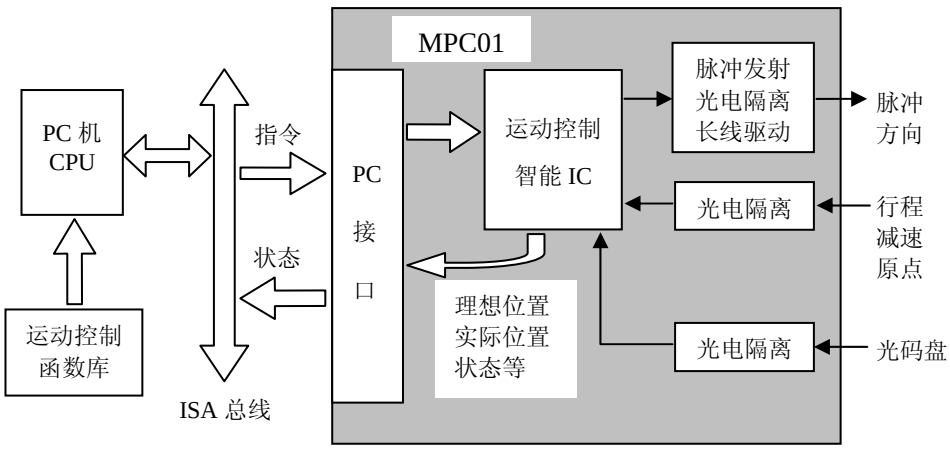
MPC01 控制卡是基于 PC 机 ISA 总线的步进电机或数字式伺服电机的上位控制单元，它与 PC 机构成主从式控制结构：PC 机负责人机交互界面的管理和控制系统的实时监控等方面的工作（例如键盘和鼠标的管理、系统状态的显示、控制指令的发送、外部信号的监控等等）；MPC01 卡完成运动控制的所有细节（包括脉冲和方向信号的输出、自动升降速的处理、原点和限位等信号的检测等等）。每块 MPC01 卡可控制 3 轴步进电机或数字式伺服电机，并支持多卡共用，以实现多于三个运动轴的控制；每轴均可输出脉冲和方向信号，以控制电机的运转；同时，可外接原点、减速、限位等开关信号，以实现回原点、保护等功能。各种开关信号由 MPC01 卡自动检测并作出反应。

MPC01 卡分为 A、B 两种类型：MPC01-A 和 MPC02-B。MPC01-A 采用更先进的控制芯片，具有 S 形升降速曲线，最高输出频率可达 2.4MHz，带有编码器反馈端口，主要适用于数字式交流伺服系统或闭环的步进电机控制系统；MPC01-B 具有梯形升降速曲线，最高输出频率可达 100KHz，主要适用于开环的步进电机控制系统。

MPC01 配备了许多功能强大、内容丰富的运动控制软件工具。MPC01 在插补算法和运动函数的执行效率方面采用了更有效的方法，提高了插补精度和速度。这些软件工具主要分为示范软件和运动函数库两大类。利用 MPC01 的示范软件既可以很快地熟悉 MPC01 控制卡的软、硬件功能，又可以方便快捷地测试执行电机及驱动系统在完成各种运动时的性能特性。MPC01 运动函数库用于二次开发，用户只要用 C／C＋＋或 Visual Basic 等编制所需的用户界面程序，并把它与 MPC01 运动库链接起来，就可以开发出自己的控制系统，例如：数控系统、检测设备、自动生产线等。MPC01 的运动函数库能够完成与运动控制有关的复杂细节（比如：升降速、直线插补、圆弧插补等），这样就可以大大缩短控制系统的开发周期。

1.2 MPC01 的结构

MPC01 控制卡作为开发运动控制系统的平台，其结构是开放式的。该卡插在 PC 机 ISA 扩展槽内使用，同时使用控制卡的数量和各卡上的控制轴数可方便地配置；MPC01 卡提供了功能强大的运动控制函数库，并可以充分利用 PC 机现有的资源来开发完美的运动控制系统。MPC01 控制卡的结构示意图如下：



DMC300 结构示意图

1.3 MPC01 的技术特性和使用范围

MPC01 控制卡主要特征有：开放式结构、使用简便、功能丰富、可靠性高等。MPC01 的特征体现在硬件和软件两个方面：在硬件方面采用 PC 机的 ISA 总线方式，适用范围广，各种设置采用简单的跳线和拨码开关；MPC01 的接线方式采用 D 型插头，可使用屏蔽线缆，并且所有的输入、输出信号均用光电隔离，提高了控制卡的可靠性和抗干扰能力；在软件方面提供了丰富的运动控制函数库，以满足不同的应用要求。用户只需根据控制系统的要求编制人机界面，并调用 MPC01 运动函数库中的指令函数，就可以开发出既满足要求又成本低廉的多轴运动控制系统。

MPC01 的技术指标主要有：

- 3 轴步进或数字式伺服控制；
- 脉冲输出速度可达 2.4Mpps（A 型）或 100KHz（B 型）；
- 梯形、S 形加减速（A 型）；
- 运动中可变速；
- 多轴高速线性插补；
- 圆弧、椭圆和螺旋插补；
- 可输出脉冲/方向或双脉冲信号；
- 每轴带有原点、减速和限位开关等接口；
- 可外接编码器位置反馈（A 型）；
- 具有外部定时器中断；
- DOS 环境下 C 语言运动函数库；
- WINDOWS 环境下 DLL 库；
- 演示和测试软件、例子程序。

正是由于 MPC01 的开放式结构，使之应用范围十分广泛，在使用步进电机和数字式伺服电机的 PC 机运动控制系统中，都可以使用 MPC01 作为核心控制单元，例如：

- 数控机床、加工中心、机器人等；
- X-Y-Z 控制台；
- 绘图仪、雕刻机、印刷机械；
- 送料装置、云台；
- 打标机、绕线机；
- 医疗设备；
- 包装机械、纺织机械；

等等。

1.4 MPC01 的运动控制功能

MPC01 控制卡的运动控制功能主要取决于运动函数库。运动函数库为单轴及多轴的步进或伺服控制提供了许多运动函数：单轴运动、多轴独立运动、多轴插补运动等等。另外，为了配合运动控制系统的开发，还提供了一些辅助函数：中断处理、编码器反馈、间隙补偿等。下面简单介绍一下这些函数的功能。

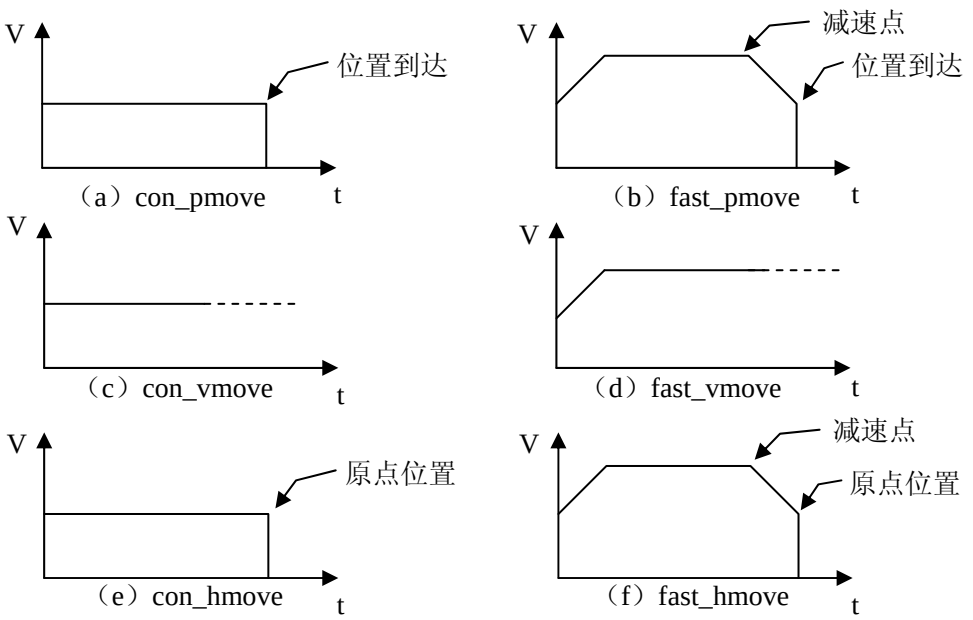
1.4.1 单轴运动控制

单轴运动有三个基本的类型：

- 点位运动（pmove）
- 连续运动（vmove）
- 回原点运动（hmove）

这些运动又可以在常速模式或梯形速度模式下工作，因此，总共有六种基本运动类型，列表如下：

con_pmove	以常速移动指定距离（图（a））
fast_pmove	以梯形速度移动指定距离（图（b））
con_vmove	以指定的常速连续运动（图（c））
fast_vmove	加速后保持在指定高速的连续运动（图（d））
con_hmove	以常速运动至原点（图（e））
fast_hmove	加速后快速移至原点位置（图（f））



运动速度图形

注：DMC300A 的 fast 类型函数，升降速可设置为 S 形。

带有升／降速控制的运动函数称之为快速（fast）运动函数，譬如：fast_pmove，fast_vmove 和 fast_hmove，而常速运动函数则称之为常速（con）运动函数，如 con_pmove，con_vmove，con_hmove。

1.4.2 多轴独立运动控制

多个运动轴能以独立的形式进行点位运动、连续运动和回原点运动（同时开始，不一定同时到达）。这类运动一般在函数名的末尾以 2 或 3 来指明参加运动的轴数。例如 con_pmove2 是一个两轴同时独立做点位运动的函数，fast_home3 是三轴独立做回原点运动的函数。这一类函数总共有 12 个。

1.4.3 多轴插补运动控制

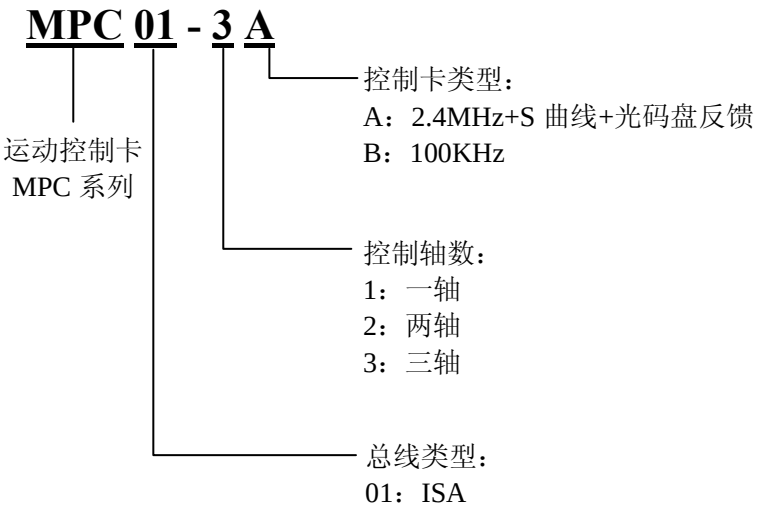
多轴插补函数能以特定的矢量速度执行线性、圆弧、椭圆和螺旋插补运动。参与插补运动的各轴同时开始运动，并且按照特定的算法同时到达各自的目标位置。线性插补函数允许两轴或三轴沿直线运动；圆弧类插补函数允许两轴沿着圆弧或椭圆运动，以及三轴作螺旋运动。做直线插补运动时，可以采用均匀矢量速度方式或梯形矢量速度方式。例如，fast_line3 函数让三轴以梯形矢量速度走直线运动。这一类的函数总共有六个。

1.4.4 中断、光码盘反馈和其它能力

MPC01 的运动函数库也提供其它多种辅助功能。例如：中断管理函数，包括处理中断服务、外部定时器中断、开／关控制卡至 PC 机的中断申请等；MPC01-A 型的控制卡上每轴都有光码盘反馈信号的接口，能对光码盘反馈信号进行自动计数，获得实际位置，尤其是在一些以步进电机作为执行机构的系统中，可以构成半闭环的控制系统，以避免步进电机的丢步或过冲等现象；并且光码盘的 Z 相信号可配合原点开关信号做精密的回原点定位；另外，还有间隙补偿函数，在机械结构存在间隙时，往复运动的位置精度会受到影响，在电机每次改变方向时应进行间隙补偿。

1.5 MPC01 型号说明

MPC01 控制卡分为 A、B 两种类型，控制轴数 1~3 轴可选。因此，其型号定义为：



2 控制卡的安装

2.1 硬件安装

MPC01 控制卡对 PC 机的硬件要求十分简单：带有 ISA 插槽的 486 以上机型即可。为了整个控制系统的可靠性，建议使用工控 PC 机。

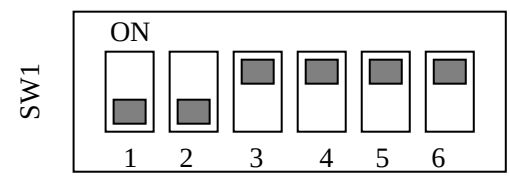
MPC01 控制卡上有两个拨码开关，用于设定控制卡的基地址和中断号。SW1 用于设置 MPC01 在 PC 总线中的基地址，出厂设置为 0X300（十六进制 300H）。SW2 和两个跳线用于设置 MPC01 的中断号，用户可选择 IRQ2、3、5、7、10、11、12、15 来中断 PC，也可以不用中断。出厂设置为“无中断”。

一般控制卡的安装步骤如下：

1．设置基地址；
2．设置中断号；
3．设置减速开关的方向；
4．将控制卡插入 PC 机。

2.1.1 设置控制卡基地址

MPC01 占用 PC 地址总线上的 16 个连续地址，其基地址（或起始地址）由拨码开关设置。控制卡的地址设置要保证其 16 个连续地址不能与 PC 机内的其它板卡的地址相冲突，否则，可能出现不可预计的现象。在多卡共用时，每块控制卡的基地址也不能相同。控制卡的基地址出厂时设置为 0X300，即 0X300~0X30F 全部由 MPC01 占用。如果用户需要改变该设置，可通过拨码开关 SW1 来重新设置。SW1 上的 6 个开关分别对应 PC 机的以下 6 根地址线（A9~A4）：



ON：0
OFF：1

地址线：A9 A8 A7 A6 A5 A4
缺省值： 1 1 0 0 0 0（0X300）

拨码开关的位置为 ON 时，相应的地址线为 0。例如：如果把基地址设置为 0X300（十六进制，其二进制数为 11 0000 0000），其中 A9~A4 为 11 0000，则拨码开关的设置为：

开关	1	2	3	4	5	6
设置	OFF	OFF	ON	ON	ON	ON

注意：改变基地址设置后，应在应用程序中调用 set_board_base 函数，以便运动函数库按照所设置的地址发送控制指令。如果设置的地址在 0X280 与 0X340 之间，可调用 auto_set 函数自动完成软件的地址设置。在多卡使用时，应设置不同的基地

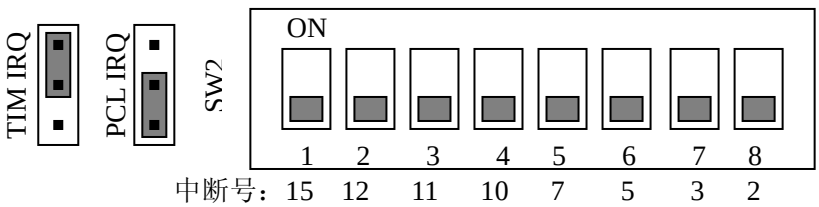
址。下表为部分地址设置：

基地址	拨码开关状态					
	1（A9）	2（A8）	3（A7）	4（A6）	5（A5）	6（A4）
0X280	OFF（1）	ON（0）	OFF（1）	ON（0）	ON（0）	ON（0）
0X290	OFF（1）	ON（0）	OFF（1）	ON（0）	ON（0）	OFF（1）
0X300*	OFF（1）	OFF（1）	ON（0）	ON（0）	ON（0）	ON（0）
0X310	OFF（1）	OFF（1）	ON（0）	ON（0）	ON（0）	OFF（1）
0X320	OFF（1）	OFF（1）	ON（0）	ON（0）	OFF（1）	ON（0）
0X330	OFF（1）	OFF（1）	ON（0）	ON（0）	OFF（1）	OFF（1）
0X340	OFF（1）	OFF（1）	ON（0）	OFF（1）	ON（0）	ON（0）

*缺省地址。

2.1.2 设置中断号

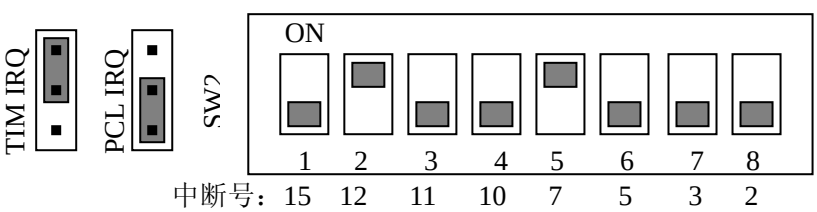
MPC01 提供两种中断：外部定时器中断和运动结束中断。中断号的设置由 SW2 和跳线 TIM IRQ、PCL IRQ 完成。TIM IRQ 用于确定外部定时器的中断号，PCL IRQ 用于确定运动结束的中断号。当跳线端子在上面时，相应的中断号由 SW2 的 1、2、3、4 选定；当跳线端子在下面时，相应的中断号由 SW2 的 5、6、7、8 选定。



SW2 的开关对应的中断号为：

开关	1	2	3	4	5	6	7	8
IRQ	15	12	11	10	7	5	3	2

例如：将外部定时器中断号设置为 IRQ12，运动结束中断号设置为 IRQ7，操作如下：



1. TIM IRQ 跳线端子在上面，并将开关 2 拨到 ON 状态，这样外部定时器中断号为 IRQ12；
2. PCL IRQ 跳线端子在下面，并将开关 5 拨到 ON 状态，这样运动结束中断号为 IRQ7。

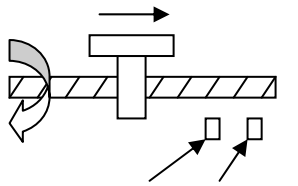
注意：

1. 跳线端子 TIM IRQ 和 PCL IRQ 不能位于同一位置；

2. 多卡共用时，不同的控制卡不能设置同一中断号，否则将不会产生中断；设置后应调用 set_board_irq 通知运动函数库；
3. MPC01 控制卡的中断号的设置不能与其它硬件的中断号冲突，否则可能会造成 PC 死机。

2.1.3 设置减速开关的方向

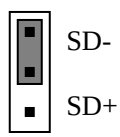
在某个轴的梯形速度（由 set_profile 设置）运动过程中，如碰到减速开关，则该轴的运动速度将自动从高速减到低速，并保持低速运行。一般情况下，减速开关与原点接近开关配合使用，以提高在高速回原点时的定位精度。将减速开关安装在原点接近开关的前面，在高速回原点时，运动机构先碰到减速开关，使之减速，并以低速靠近原点接近开关，在到达原点位置时，停止运动。如果没有减速开关，在高速回原点时，碰到原点接近开关将立即停车，由于运动机构的惯性、原点接近开关的有效工作范围等因素，将会降低回原点的定位精度，并可能给机械造成冲击。



根据减速开关的安装位置，应设置 MPC01 控制卡上减速开关的方向：

当跳线位于上方时，减速开关应安装在负向；

当跳线位于下方时，减速开关应安装在正向。



注意：这里的正、负向是指控制卡发送脉冲的方向，可能与运动机构的实际运动方向并不一致。

2.1.4 将控制卡插入 PC 机

为了保证安全，插卡时应下列步骤操作：

1．关 PC 机，并切断电源；
2．打开 PC 机箱，选择未用的扩展槽，并插入 MPC01 控制卡；
3．固定 MPC01 控制卡，并盖好 PC 机；
4．连接 MPC01 与电机驱动器等；
5．接上电源，并启动 PC 机。

2.2 软件安装

MPC01 控制卡对操作系统的要求也十分简单：DOS6.0 及以上版本或 WINDOWS9X 均可。用户可根据自己的软件技术优势进行选择。

MPC01 配有 DOS 环境和 WINDOWS 环境下的运动函数库和演示软件，以满足不同环境下的运动控制系统的开发和测试。MPC01 软件的安装非常简便，只需运行安装程序，即可将相应的软件安装到硬盘上。

将 MPC01 卡的软件安装到硬盘的步骤如下：

1. 将 MPC01 卡软件安装盘插入 A 盘驱动器（1.44Mb）；

2. 在 DOS 提示符下输入 A:\INSTALL，并回车；
3. 所有的 MPC01 软件将自动地安装在 C:\DMC300 目录下。

在 C:\DMC300 目录下，共有三个子目录：DEMO、DLL、LIB。

DEMO 目录中为 MPC01 卡的 DOS 环境下的演示软件，该演示软件为中文界面，操作十分简单。利用该软件可以方便地改变每个控制轴的速度、加速度和运动距离等，以完成对 MPC01 卡、电机和驱动器的性能测试等。

DLL 目录下为 MPC01 卡的动态链接库，用于 WINDOWS 环境下的控制系统的开发，并且还包含了用于 VB 开发的模块 DMC300.BAS，以及用 VB 开发的示例程序，参照示例程序，用户能够很快地掌握 DLL 库的调用方法。

LIB 目录中为 MPC01 卡的 C 语言运动控制函数库及相应的头文件，用于 DOS 环境下控制系统开发。同时该目录中也给出了简单的示例程序。

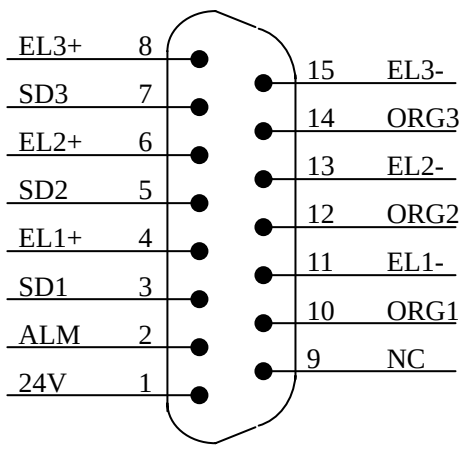
3 MPC01 接口

MPC01 控制卡的接口为 15 芯（DB15）和 25 芯（DB25）的两个 D 型插头。DB15 为三个轴的开关量信号（原点、减速、限位等），采用 12~24DCV 光电隔离；DB25 为三个轴的脉冲量信号（脉冲、方向、编码器反馈等），采用 5DCV 高速光电隔离。外接线可采用屏蔽线缆，以提高控制卡的抗干扰能力。

3.1 开关量信号接口

3.1.1 DB15 接口定义

DB15 的接口图如下：



DB15 的接口定义为：

编号	定义	编号	定义
8	限位开关 3+	15	限位开关 3-
7	减速开关 3	14	原点开关 3
6	限位开关 2+	13	限位开关 2-
5	减速开关 2	12	原点开关 2
4	限位开关 1+	11	限位开关 1-
3	减速开关 1	10	原点开关 1
2	外部报警（仅对 MPC01-A）	9	未用
1	光隔电源（12~24DCV）		

各引脚功能如下：

- 引脚 1（24V）为 12~24DCV 的光电隔离电源，该电源由外部提供。
- 引脚 3、5、7（SD1、SD2、SD3）输入减速开关信号，可根据控制卡上的跳线

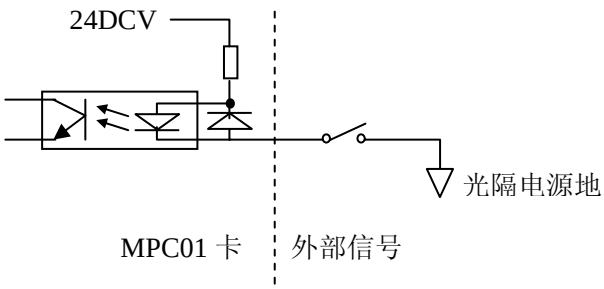
选择是正向运转时减速，还是负向运转时减速；

- 引脚 10、12、14（ORG1、ORG2、ORG3）输入原点开关信号；
- 引脚 4、6、8（EL1+、EL2+、EL3+）和 11、13、15（EL1-、EL2-、EL3-）输入行程两端的限位开关信号。
- 引脚 2（ALM）为外部报警信号，当控制卡接收到该信号，该卡的各控制轴将立即停止发送脉冲。该信号只对 MPC01A 型控制卡有效。

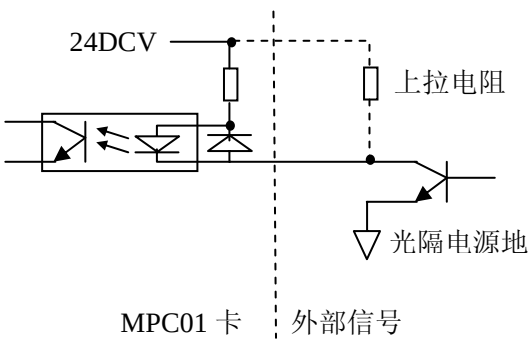
3.1.2 DB15 接线方法

DB15 接口输入均为开关量信号，其输入信号可以是触点型开关，也可以是 NPN 输出的传感器接近开关等。

1．DB15 与触点型开关的接线方法



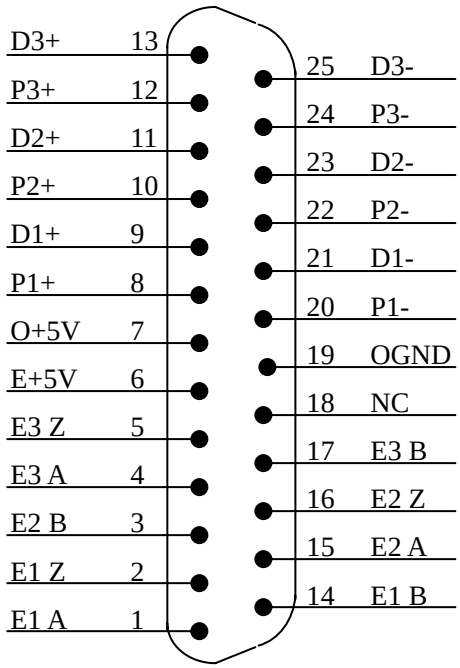
2．DB15 与 NPN 输出的传感器接近开关的接线方法



3.2 脉冲量信号接口

3.2.1 DB25 接口定义

DB25 的接口图如下：



DB25 的接口定义为：

编号	定义	编号	定义
13	方向 3+	25	方向 3-
12	脉冲 3+	24	脉冲 3-
11	方向 2+	23	方向 2-
10	脉冲 2+	22	脉冲 2-
9	方向 1+	21	方向 1-
8	脉冲 1+	20	脉冲 1-
7	光隔电源 5DCV1	19	光隔离地 1
6	光隔电源 5DCV2	18	未用
5	光码盘 3 Z	17	光码盘 3 B
4	光码盘 3 A	16	光码盘 2 Z
3	光码盘 2 B	15	光码盘 2 A
2	光码盘 1 Z	14	光码盘 1 B
1	光码盘 1 A		

各引脚功能如下：

- 引脚 7（O+5V）为脉冲和方向信号的光电隔离电源（+5DCV）的正端，引脚 19

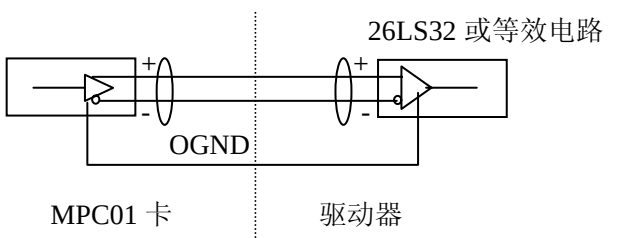
（OGND）为该电源的地。该电源由外部提供，供三个轴的脉冲和方向信号的光电隔离使用；

- 引脚 6（E+5V）为光码盘的光电隔离电源（一般为+5DCV，如果使用高于 5V 的隔离电源，定货时应特别说明）的正端，供三个轴的光码盘及其隔离使用。当光码盘的供电为+5V 时，可与引脚 7（O+5V）共用；如果光码盘与光电隔离的电源不同，应注意共地；
- 引脚 8（P1+）和 20（P1-）、10（P2+）和 22（P2-）、12（P3+）和 24（P3-）分别输出三个轴的差分式脉冲信号，引脚 9（D1+）和 21（D1-）、11（D2+）和 23（D2-）、13（D3+）和 25（D3-）分别输出三个轴的差分式方向信号。对于仅需要单端式信号的驱动器，只要接该差分信号的正端即可（参见下图）；对于接收双脉冲信号的驱动器，P 端为正转（CW）脉冲输出端，D 端为反转（CCW）脉冲输出端（这种情况下，应调用 set_output_mode 设置 MPC01 卡的脉冲输出模式，参见 set_output_mode 函数说明）；
- 引脚 1~5、14~17 分别输入三个轴光码盘反馈信号，第一轴：E1 A、E1 B、E1 Z，第二轴：E2 A、E2 B、E2 Z，第三轴：E3 A、E3 B、E3 Z；仅 MPC01-A 型卡有此接口。

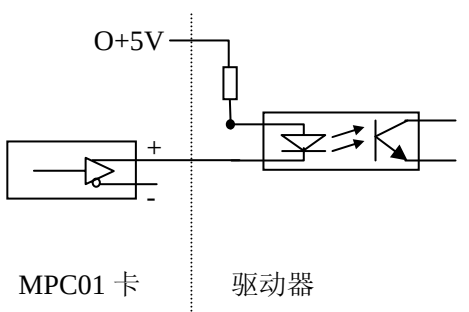
3.2.2 DB25 接线方法

DB25 为 MPC01 的脉冲量信号接口，包括脉冲和方向的输出信号、编码器的反馈信号，其接线方法分别如下：

1．DB25 脉冲和方向输出接线方法

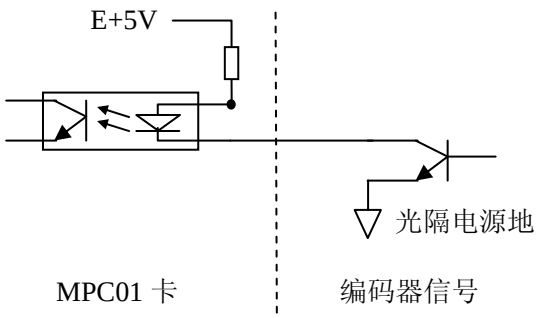


差分信号接线方法



单脉冲信号接线方法

2．DB25 编码器反馈信号的接线方法



4 运动控制系统的开发

4.1 开发 DOS 环境下的运动控制系统

利用 MPC01 运动函数库可以直接开发 DOS 环境下的运动控制系统。MPC01 的 C 语言运动函数库能够完全处理运动控制方面的工作，因此开发者可以集中精力来设计完美的人机界面，编制界面方面的程序（通常情况下，这方面的编程工作比处理底层运动控制的工作量要少得多）。开发 DOS 环境下的运动控制系统，只需要一个 C 语言编译器。MPC01 的 C 语言运动函数库是用 Turbo C 编译制作的。开发者可以用 Turbo C2.0、3.0 或 Borland C++2.0、3.0、3.1 或更高的版本来与 MPC01 运动函数库相链接。

4.1.1 链接运动函数库

MPC01 运动函数库的链接非常简单，首先在需要调用运动函数的程序模块中包含头文件 dmc300.h，然后就可以调用库中的任意运动函数。程序编辑完后，编译这个程序并将它与运动函数相链接。可以在集成环境下（以.mak 文件或.prj 文件的方式）完成这项工作，也可以用行命令来完成。

需要特别注意的是：编译和链接工作一定要在大模式（Large memory model）下进行。这是因为 MPC01 运动函数库是在大模式下产生的。

对于 MPC01 的 DOS 环境下 C 语言运动函数库，若用户使用 Borland C++开发环境，应该将头文件中注释掉的 extern "C"{ }恢复，以便能够正确编译链接。若使用 Trubo C 开发环境，则无需如此。

4.1.2 简单的定位运动例程

下面以一个简单的例子来说明怎样开发运动控制程序。

让第一轴电机以下列梯形速度参数运动 5000 步：

起始速度（ls）：400pps

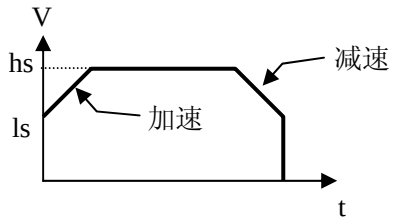
高速(hs)：6000pps

加／减速度：10000ppss

速度和加减速度单位如下：

pps=每秒脉冲数(pulse per second)；

ppss=每秒 pps（pps per second）；



那么该运动控制程序如下：

```
/*pmovel.c  让轴 1 电机运动 5000 步*/
#include “dmc300.h”
void main ()
{
        set_board_num(1); /*设置控制卡数目为 1*/
        set_axe(1,1); /*卡号为 1 的控制卡上有一个轴*/
        set_board_base(1,0x310); /*设置第一卡的基地址为 0x310*/
        init_board(); /*初始化控制卡*/
        set_profile(1,400,6000,10000); /*设置第一轴梯形速度曲线*/
        fast_pmove(1,5000); /*第一轴快速移动 5000 个脉冲的距离*/
        return;
}
```

请注意，如果调用 fast_pmove 函数，那么应先调用 set_profile 函数设置所需要的梯形速度，否则，电机运转将使用缺省参数：set_profile（axis, 200, 800,10000）。对于 con_pmove 运动，应先调用 set_conspeed 函数设置常速，否则 conspeed 的缺省值是 200pps。

4.1.3 简单的连续运动和回原点运动

连续运动函数使电机按照一个特定的速度一直运转，直到调用 sudden_stop 或 decel_stop 使其停止，或者遇到限位信号、外部报警信号等。这些函数被称之为 vmove 函数，是因为电机以一特定的速度(velocity)运动。下面的程序为连续运动的例程，执行时要求输入一个常速度，并以此速度作连续运动，直到按任意键停止：

```
/*vmove..c  文件用来测试 vmove 运动*/
#include <stdio.h>
#include <conio.h>
#include “dmc300.h”

void main()
{
        int dir;
        double con_speed;

        set_board_num(1); /*设置控制卡数目为 1*/
        set_axe(1,1); /*卡号为 1 的控制卡上有一个轴*/
```

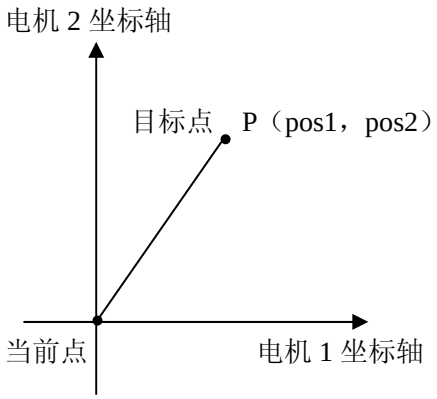
```
        init_board();/*初始化控制卡*/
        printf("\n Please input a speed of con_vmove \n");
        scanf("%f, &con_speed);
        printf("\n Please input a direction of vmove \n(1 for positive while -1 for the
        negative)\n");
        scanf("%d",&dir);
        printf("\n Hit enter to start motion:\n");
        getch();

        set_conspeed (1,con_speed);
        con_vmove(1,dir); /*令第一轴以常速连续运转*/
        printf('\n Press any key to stop vmove motion");
        getch();
        sudden_stop(1); /*急停指令*/
        return;
}
```

调用回原点运动函数可以使机床或运动台返回原点，这些函数以常速或梯形速度运动，直到 MPC01 卡接收到相应的原点接近开关发出的信号为止。下面是一个调用 con_hmove 的例子，达到原点时运动将自动停下来。

```
/*vmovel.c  文件用来测试 vmove 运动*/
/*hmovel.c:file for testing hmove*/
#include <stdio.h>
#include <conio.h>
#include "dmc300.h"

void main()
{
        int dir;
        double con_speed;

        auto_set(); /*自动检测控制卡并自动设置*/
        init_board();
        printf('\n Please input a speed of con_hmove\n");
        scanf("%f",&con_speed);
        printf("\please input a direction of hmove\n(1 for positive while-1 for
        negative)\n");
        scanf("%d",&dir);
        printf("\n Hit enter to start motion:\n");
        getch();

        set_conspeed (1, con_speed);
        con_hmove(1,dir)
        wait_done(1);
        return;
}
```

4.1.4 多轴插补运动举例

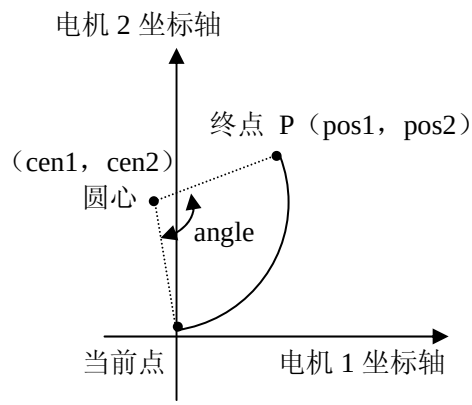
多轴插补运动包括线性、圆弧和螺旋线运动，它们的运动速度由矢量速度（常矢量速度或梯形矢量速度）决定。下面是一个简单的直线运动的例子：




```
/*line.c 演示两个轴以梯形速度走直线运动*/
#include”dmc300.h”
void main()
{
        set_board_num(1);
        set_axe(1,2);
        init_board(); /*初始化控制卡*/
        set_vector_profile(600,3000,10000);/*设置矢量梯形速度*/
        fast_line2(1,5000,2,10000); /*轴 1 移动 5000 个脉冲，轴 2 移动 10000 个脉
        冲，各轴的速度为梯形矢量速度在各轴上的分量*/
        return;
}
```

注意：直线运动可以分为两类：常速模式（con_line）和快速模式(fast_line)。上述例子演示的就是一个以梯形速度走直线运动的例子。在这种模式下，矢量低速、矢量高速和矢量加速度应在调用前给定，否则这些参数将取缺省值。在常模式下，只需设置常矢量速度。

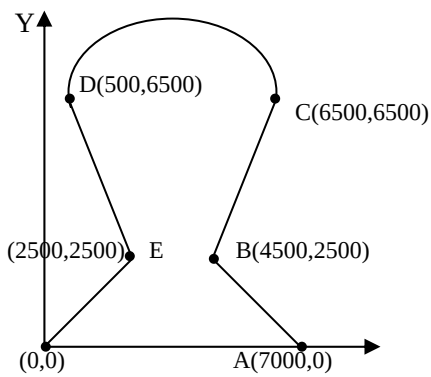
下面是一个简单的圆弧插补的例子：



```
   /*arcl.c  让两轴做圆弧运动*/
#include<stdio.h>
#include<conio.h>
#include”dmc300.h”

void main()
{
        int dir;
        set_board_num(1);
        set_axe(1,2);
        init_board();
        set_vector_conspeed(800);/*设置常矢量速度*/
        arc_center(1,2,1000,-2000,180); /*以（1000，-2000）为圆心顺时针插补 180
        度*/
        return;
}
```

下面再来看一个直线和圆弧运动组成的连续运动例子，运行这个程序将驱动运动轴 1 和轴 2 走一封闭曲线。

```
/*multi.c demostrates a continuous motion composed of lines and arcs*/
#include<stdio.h>
#include<conio.h>
#include “dmc300.h”

void multi_moves(int ch1, int ch2);
```




```
void main()
{
        set_board_num(1);
        set_axe(1,2);
        init_board();.
        set_vector_conspeed(600);
        set_vector_profile(600, 4000, 10000);
        printf(“press any key to start the motion!”);
```

```
        getch();
        multi_moves(CH1,CH2);
        return;
}
void multi_moves(int ch1,int ch2)
{
        fast_line2(ch1,7000,ch2,0);
        con_line2(ch1,-2500,ch2, 2500);
        con_line2(ch1, 2000,ch2,4000);
        arc_final(ch1, ch2, CCW,-6000,0,3300);
        con_line2 (ch1,-2500,ch2, -2500);
        wait_done(ch1);
        wait_done(ch2);

        return;
}
```

4.1.5 中断产生及处理

处理中断通常是一项极具挑战性的工作。但如果使用 MPC01 所附带的库函数来处理中断，这一工作将变得相当简单。

下列示意图说明中断管理的机制：

INTR：运动轴中断源；

IRQ：控制卡向 PC 申请中断；

ISR：中断服务程序。

控制卡的每个轴在运动完成时都可以产生中断信号，这个中断信号使控制卡向 PC 发出中断申请（IRQ），然后 PC 执行一个中断服务程序，以响应这个中断。MPC01 运动函数库提供了一些函数来处理中断：

管理轴中断信号（INTR）：

int enable_axis_intr(int ch);/*使一个轴可以产生中断*/

int disable_axis_intr(int ch);/*使一个轴不能产生中断*/

管理板中断信号(IRQ)：

int set_board_irq(int ch, int irq_number):/*设置 MPC01 板卡的中断号*/

int enable_board_irq();/*使控制卡能向 PC 机申请中断*/

管理 PC 中断服务(ISR)：

int set_board_isr(int(*my_isr)());//让一个常规函数执行控制卡的中断服务//

管理 DOS Timer 中为(IRQ 8);

int enable_timer_irq();/*使 DOS timer 可以中断 PC 机*/

```
int disable_timer_irq();/*使 DOS timer 不能中断 PC 机*/
int set_timer_isr(int(*my_isr)());/*让一个常规函数执行 DOS Timer 的中断服务*/

/*intrl.c 演示处理第二轴的中断*/
#include<conio.h>
#include<stdio.h>
#include"dmc300.h"

#define BEEP()_asm{\
        _asm sub bx, bx /*Page03    */\
        _asm mov ax, 0E07h/*TTY bell    */\
        _asm int 10h /*BIOS10    */\
}
int my _isr(void);
int main()
{
        unsigned nIRQ_no=2; /*IRQ line number*/
        set_board_num(1);
        set_axe(1,2);
        init_board();
        set_board_irq(1,nIRQ_no);
        set_board_isr(my_isr);/*the function'my_isr'dealing with interrupt */
        enable_board_irq();
        enable_axis_intr(2);
        printf("\n prss any key to start a 'con_pmove' motion");
        getch();
        con_pmove(2,-1000);
        wait_done(2);
        disable_board_irq();
        return 0;
}
int my_isr()
{
        int i ;
        for(i=0;i<10;i++)BEEP();
        return 0;
}
```

在这个例子中，一个用户编写的常规函数钩住中断，处理中断事务，充当中断服务程序的角色，所以即使开发者对中断了解不深，仍然可以以一种极为简单的方式来处理控制卡产生的中断。

4.2 开发 Windows 下的运动控制系统

利用 MPC01 的动态链接库（DLL），开发者可以很快开发出 Windows9X 下的运动控制系统。这个动态链接库（32 位）是由 Microsoft Visual C++5.0 生成，它可以用 Visual Basic 5.0 版或更高版本、Visual C++5.0 版或更高版本来调用，生成的运动控制程序可以在 Windows9X 下运行。

4.2.1 开发 Visual Basic 控制程序

为了开发基于 Windows 的运动控制程序，用户可以使用 VB5.0 或更高版本，开发过程与 DOS 下用 C 开发很类似。步骤如下：

1．用 Visual Basic 写一个界面程序；

2．将 DMC300.bas 文件添加到 VB Project 中去（与 C 语言里包含头文件类似）；

3．把动态链接库 DMC300.DLL 拷入 c:\windows\system 子目录；

4．在应用程序中调用运动函数。

所有 Visual Basic 的教材都介绍了如何写介面程序，包括按钮、对话框以及菜单等。对于熟悉 Visual Basic 和 MPC01 运动函数库的开发者来说，一个由输入框和命令按钮组成的基于 Windows 的简单运动程序，可以在几分钟内就可以开发。当然，开发一个复杂的运动应用程序所需的时间可能会长一些。

4.2.2 调用动态链接库函数

调用 DLL 中提供的运动函数是一件十分容易的事情。由于 dmc300.bas 文件包含有 DLL 函数的必要说明：程序中调用 DLL 函数与 C 中调用基本一样。例如：如果让轴 1 运动 4000 步，调用语句为：

```
temp=fast_pmove(1,4000)
```

其中 temp 是一整数变量。

如果调用 viod 类型的运动函数，如 suden_stop，调用语句为：

```
Call sudden_stop(1)        这里的 1 表示第一轴
```

4.2.3 动态链接库中的函数

绝大多数 DOS 环境下的函数都包括在 DLL 动态链接库中，但并非是所有的函数。例如：由于 WINDOWS 环境下的中断处理较为复杂和繁琐，所以与中断相关的函数还没有包括进来；WINDOWS 环境下和 DOS 环境下对键盘、显示的管理也大有不同，因此与此相关的函数（如 wait_done 等）也没有包括在 DLL 库中。但对 WINDOWS 消息传递机制比较熟悉的开发者，一般都能编制出相同功能的 WINDOWS 函数。

5 函数描述

本章详细地描述了 MPC01 运动库中的每一个函数。其中，在函数库中使用的单位和函数返回值约定通常如下：

单位

- 位移（或距离）的单位为 P（Pulse），即脉冲数；
- 速度的单位是 PPS（Pulse/sec），即脉冲/秒；
- 加速度和减速度的单位是 PPSS（Pulse/sec^2），即脉冲 / 秒 ^{2}。

函数返回值

运动库中的大多数函数是整型函数，它们的返回意义如下：

0　　函数执行正确；

-1　　函数执行错误。

5.1 控制卡和轴设置函数

该类函数主要用于设置 MPC01 卡的使用数量、每块 MPC01 卡的基地址、中断号、控制轴数以及每轴的输出模式，速度、加速度、S 形曲线等的设置和读取等等。相关函数有：

int set_board_num（int num）；/*设置一台 PC 机上使用控制卡的数量*/

int set_board_base（int board_no，int base）；/*设置每块控制卡的基地址*/

int get_board_base（int board_no）；/*读取每块控制卡的基地址*/

int set_axe（int board_no，int num）；/*设置每块控制卡上的轴数*/

int auto_set(void)；/*自动检测和自动设置控制卡*/

int init_board（void）；/*对控制卡硬件和软件初始化*/

int get_max_axe（void）；/*读取总轴数*/

int set_ board_irq（int board_no，int IRQ_no）；/*设置每块控制卡的中断号*/

int get_board_irq（int board_no）；/*读取每块控制卡的中断号*/

int set_outmode（int ch，int mode，int outlogic）；/*设置各轴输出模式*/

int set_home_mode（int ch，int home_mode）；/*设置回原点模式*/

int set_conspeed（int ch，double conspeed）；/*设置各轴常速度*/

double get_conspeed（int ch）；/*读取各轴常速度*/

int set_profile（int ch，double ls，double hs，double acc）；/*设置梯形速度*/

int get_profile（int ch，double &ls，double &hs，double &acc）；/*读取各轴梯形速度*/

double get_rate（int ch）；/*读取当前运转速度*/

int set_s_curve（int ch，int mode）；/*设置 S 形曲线是否有效*/

int set_s_section（int ch，double rate）；/*设置 S 形曲线段*/

函 数 名：set_board_num

目　　的：用于设置在同一台 PC 机上所使用的 MPC01 控制卡的数量。

语　　法：void set_board_num（int num）；/*num 为所使用的控制卡数量*/
调用例子：set_board_num（3）；/*在该 PC 机上有 3 块 MPC01 卡*/
描　　述：每块 MPC01 控制卡最多可控制三个轴的运动，为满足多于三轴的控制系统，可同时使用多个 MPC01 卡。例如：对于一个 8 轴控制系统，可使用三块 MPC01 卡：两块 MPC01-3 和一块 MPC01-2。用函数 set_board_num 可以设置控制卡的数量（缺省值为 1）。注意：设置的卡数应与实际插卡数一致。
返 回 值：成功返回 0，失败返回-1。
系　　统：DOS，WINDOWS
参　　见：

函 数 名：set_board_base，get_board_base
目　　的：用 set_board_base 函数来设置每块 MPC01 控制卡的基地址。该函数应在 init_board 函数前调用，否则无效。用 get_board_base 函数可读取每块 MPC01 已设置的基地址值，该函数应在 set_board_base 之后调用。
语　　法：int set_board_base（int board_no，int base）；
int get_boardd_base（int board_no）；
board_no：卡号；
base：所设置的基地址；
调用例子：set_board_base（1，0x310）；/*将第 1 块 MPC01 的基地址设置为 x310*/
base_addr=get_board_base（2）；/*读取第 2 块 MPC01 的基地址并赋给变量 base_addr。*/
描　　述：MPC01 控制卡的基地址出厂设置为 0x300，如果要改变 MPC01 控制卡的基地址，首先要改变拨码开关 SW1 的设置（参见 MPC01 控制卡基地址设置方法），然后在 init_board 前调用 set_board_base 函数（仅需调用一次），否则运动函数不能与控制卡通讯。注意：在仅使用一块 MPC01 控制卡时，board_no 应为 1；在使用多块 MPC01 时，应分别设置各卡的基地址，并且基地址的软、硬件设置应保持一致。在没有先调用 set_board_base 的情况下，调用 get_board_base 函数，其返回值将为缺省值。如果 set_board_base 在前面已调用，则 get_board_base 将返回 set_board_base 所设置的基地值。
返 回 值：如果调用成功，set_board_base 返回 0 值，否则返回-1。get_board_base 返回卡号为 board_no 的控制卡的基地址。
系　　统：DOS，WINDOWS
参　　见：init_board/*自动设置控制卡基地址*/

函 数 名：set_axe，get_max_axe
目　　的：set_axe 用于设置每块 MPC01 控制卡的控制轴数，get_max_axe 用于读取总的控制轴数。
语　　法：int set_axe（int board_no，int num）；
int get_max_axe（void）；
board_no：卡号；
num：所设置的轴数。

调用例子：set_axe（1，2）；/*第 1 块 MPC01 卡的控制轴数设置为 2。*/
max_axe_num=get_max_axe（）；
描　　述：在多于 3 轴的控制系统中，应使用多块 MPC01 卡，并需调用 set_axe 设置各卡上的轴数，使运动函数库能正确识别各控制轴号。注意：set_axe 设置的轴数应与相应卡号上的实际轴数一致。然后用 get_max_axe 函数可读取所有 MPC01 卡的总控制轴数。
返 回 值：如果调用 set_axe 成功返回 0，否则返回-1。get_max_axe 返回总控制轴数。
系　　统：DOS，WINDOWS
参　　见：set_board_num

函 数 名：auto_set
目　　的：用 auto_set 函数自动检测 MPC01 卡的数量、基地址、各卡上的轴数，并自动设置每块 MPC01 控制卡。
语　　法：int auto_set（void）；
调用例子：auto_set（）；/*自动检测和自动设置 MPC01 卡*/
描　　述：当所使用的 MPC01 卡的基地址设置在 0x280~0x340 之间时，可以调用 auto_set 完成 MPC01 卡的数量、各卡的基地址和轴数的自动检测，并自动设置这些参数。auto_set 的功能相当于连续调用 set_board_num、set_board_base、set_axe 的功能。注意：此时卡号和轴号的顺序由各卡上的基地址设置顺序确定。
返 回 值：如果调用成功，auto_set 函数返回总轴数，否则返回 0。
系　　统：DOS，WINDOWS
注　　释：当使用 auto_set 进行自动检测和设置时，各 MPC01 卡的基地址硬件设置应在 0x280~0x340 之间，否则将检测不到。
参　　见：set_board_num, set_board_base, set_axe

函 数 名：init_board
目　　的：用 init_board 函数初始化控制卡。
语　　法：int init_board（void）；
调用例子：init_board（）；
描　　述：在 MPC01 卡的数量、各控制卡的基地址和轴数，或者用 auto_set 自动检测和设置之后，必须调用 init_board 函数来对控制卡进行初始化。init_board 函数主要初始化控制卡的各个寄存器、各轴的脉冲输出模式（脉冲/方向）、常速度（2000pps）、梯形速度（初速 2000pps，高速 8000pps，加减速 80000ppss）、矢量常速度（2000pps）、矢量梯形速度（初速 2000pps，高速 8000pps，加减速 80000ppss）等等。
返 回 值：如果调用成功，init_board 函数返回 0，否则返回-1。
系　　统：DOS，WINDOWS
注　　释：如果不调用 init_board 函数初始化，控制卡将不能正常工作。若需改变脉冲输出模式、速度等初始化数据，可调用其它函数来修改。
参　　见：set_board_num, set_board_base, set_axe, auto_set

函 数 名：set_board_irq，get_board_irq

目　　的：用 set_board_irq 函数设置每块 MPC01 控制卡的中断号（IRQ）；
用 get_board_irq 函数读取每块 MPC01 卡已设置的中断号；

语　　法：int board_irq（int board_no，int IRQ_no）；
int get_board_irq（int board_no）；
board_no：卡号；
IRQ_no：中断号；

调用例子：set_board_irq（1，2）；/*将第 1 块 MPC01 卡的中断号设为 IRQ2。*/
irq_number=get_board_irq（1）；/*读取第 1 块 MPC01 卡的中断号。*/

描　　述：每块 MPC01 卡上的任意一根控制轴在完成运动时，都可以向 PC 机产生中断（参见 enable_board_irq）。函数 set_board_irq 用于设置每块 MPC01 控制卡的中断号（IRQ），则该控制卡上的轴通过所设置的中断号向 PC 机发出中断请求。这个 IRQ 号必须与控制卡上拨码开关 SW2 的设置一致。在多卡共用时，各卡的中断号不能相同。函数 get_board_irq 用来读取 set_board_irq 函数所设置的中断号。如果 set_board_irq 没有在 get_board_irq 前调用，读取的 IRQ 号为缺省中断号。

返 回 值：如果调用成功，set_board_irq 函数返回 0 值，否则返回-1。
如果调用成功，get_board_irq 函数值返回 IRQ 号，否则返回-1。

系　　统：DOS

注　　释：中断号 IRQ 由控制卡上的拨码开关 SW2 设置。此外，控制卡的 IRQ 不能与 PC 机上已被其它设备占用的 IRQ 号重复，否则会相互干扰，出现不可预测的现象。

参　　见：enable_board_irq，set_axis_intr

函 数 名：set_outmode

目　　的：用于设置每个轴的脉冲输出模式以及信号逻辑定义。如果驱动器要求双脉冲（正向脉冲、反向脉冲）控制信号接口，或者需要改变输出信号的逻辑，那么应在 init_board 函数后调用该函数。

语　　法：int set_outmode（int ch，int mode，int outlogic）；
ch：所设置输出方式的控制轴；
mode：脉冲输出模式设置（1 为脉冲 / 方向方式，0 为双脉冲方式）；
outlogic：脉冲输出和方向输出的逻辑设置：0 为负逻辑，1 为正逻辑。

调用例子：set_outmode（2，0，1）；/*将第 2 轴的脉冲输出模式设置为双脉冲模式，负逻辑有效。*/

描　　述：在缺省情况下，init_board 函数将所有轴设置为脉冲 / 方向模式，输出信号为负逻辑。如果驱动器要求双脉冲（正向脉冲和反向脉冲）模式的输入或正逻辑，那么应在 init_board 函数后调用 set_outmode 重新设置所要求的模式。注意：控制卡的输出模式应与所连接的驱动器的输入信号模式一致，否则电机将不能正常工作。

返 回 值：如果输出方式设置成功，则 set_outmode 返回值为 0，否则返回-1。

系　　统：DOS，WINDOWS

参　　见：init_board

函 数 名：set_home_mode

目　　的：用于设置各轴回原点运动时检测原点信号的方式。

语　　法：int set_home_mode（int ch，int home_mode）

ch：是所设置的轴；

home_mode：回原点运动时检测原点信号的方式（0：仅检测原点接近开关信号，1：检测原点接近开关信号和电机上光电编码器 Z 相脉冲信号同时出现）。

调用例子：set_home_mode（1，1）；

描　　述：在被控设备（比如数控机床等）回原点运动时，MPC01 卡将自动检测原点信号，并在到达原点位置时自动停止运动。原点信号一般由接近开关发送。但在一些回原点时定位精度要求较高的场合，原点信号除了接近开关信号之外，还要检测执行电机上光电编码器的 Z 相脉冲，即仅当接近开关信号和 Z 相脉冲信号同时出现时，才表明已到达原点。函数 set_home_mode 就是用于设置每个轴在回原点运动时检测原点信号的方式。当 home_mode=0 时，仅将原点接近开关信号作为原点信号；当 home_mode=1 时，将原点接近开关信号和 Z 相脉冲信号两者同时出现作为原点信号。注意：只有执行电机上装有光电编码器时，才能将原点接近开关信号和 Z 相脉冲信号同时出现设置为回原点运动的检测信号，否则将无法正确完成回原点运动。

返 回 值：如果设置成功，返回 0，否则返回-1。

系　　统：DOS，WINDOWS

参　　见：con_home，fast_home

函 数 名：set_maxspeed

目　　的：用于设置每个轴的最大速度。

语　　法：int set_maxspeed（int ch，double speed）；

ch：所设置的控制轴；

speed：设置的最大速度值，单位为脉冲 / 秒（pps）。；

调用例子：set_maxspeed（2，10000）；/*将第 2 轴的最大速度设置为 10000pps。*/

描　　述：在缺省情况下，init_board 函数将所有轴按卡的型号设置其最大速度，即 A 型卡为 2400000，B 型卡为 100000。使用时可按照实际输出速度进行设置以获得比较好的速度精度。

返 回 值：如果输出方式设置成功，则 set_ maxspeed 返回值为 0，否则返回-1。

系　　统：DOS，WINDOWS

参　　见：set_conspeed

函 数 名：set_conspeed，get_conspeed

目　　的：用 set_conspeed 函数来设置一个轴在常速运动时的速度。

用 get_conspeed 函数来获取某个轴所设置的常速度。

语　　法：int set_conspeed（int ch，double conspeed）；

double get_conspeed（int ch）；

ch：函数所操作的轴；

conspeed：设定的常速度值，单位为脉冲 / 秒（pps）。

调用例子：set_conspeed（2，400）；
speed=get_conspeed（2）；
描　　述：函数 set_conspeed 可以设定在常速运动方式下的速度。如果多次调用这个函数，最后一次设定的值有效，而且在下一次改变之前，一直保持有效。
返 回 值：如果常速度值设置成功，set_conspeed 返回 0 值，出错时返回-1。
函数 get_conspeed 返回指定轴的常速度值。
系　　统：DOS，WINDOWS
注　　释：常速度值一般设置较低，以免造成控制电机（尤其是开环的步进电机）丢步或过冲。如果需要高速运动，最好使用梯形速度或 S 形速度方式。
参　　见：set_profile，set_vector_conspeed

函 数 名：set_profile，get_profile
目　　的：用 set_profile 函数来设定在快速运动（包括 fast_hmove，fast_vmove，fast_pmove 等）方式下的梯形速度的各参数值；用 get_profile 来读取梯形速度的各参数值。
语　　法：int set_profile（int ch，double ls，double hs，double accel）；
ch：设定的轴数；
ls：设定低速（起始速度）的值；单位为 pps（脉冲 / 秒）；
hs：设定高速（恒速段）的速度值；单位为 pps（脉冲秒）；
accel：设定加速度大小；单位为 ppss（脉冲 / 秒 / 秒）；
int get_profile（int ch，int *ls，int *hs，long*accel）
double *ls:指定起始速度的指针；
double *hs:指向高速的指针；
double *accel：指向加速度值的指针。
调用例子：set_profile（3，600，6000，10000）；
get_profile，（3，&ls，&hs，&accel）；
描　　述：函数 set_profile 设定一个轴在快速运动方式下的低速（起始速度）、高速、加 / 减速度值（减速度值等于加速度值）。这几个参数的缺省值分别为 200、800、8000。函数 get_profile 通过指针返回一个轴的梯形速度的低速、高速和加 / 减速度值。
返 回 值：如果设定参数值成功，set_profile 返回 0，否则返回-1。
如果调用成功，get_profile 返回 0 值，否则返回-1。
系　　统：DOS，WINDOWS
参　　见：set_conspeed，set_vector_conspeed，set_vector_profile

函 数 名：get_rate
目　　的：用 get_rate 函数来获取某个轴当前正在运行的速度。
语　　法：double get_rate（int ch）；
ch：函数所操作的轴；
调用例子：speed=get_rate（2）；
描　　述：函数 get_rate 可以在被控轴运动过程中读取该轴当前的运行速度。

返 回 值：函数 get_rate 返回指定轴的当前运行速度，单位：每秒脉冲数（pps）。
系　　统：DOS，WINDOWS
参　　见：get_profile

函 数 名：set_s_curve，set_s_section
目　　的：用 set_s_curve 函数来设置某个轴在快速运行时是否采用 S 形升降速；
用 set_s_section 函数设置 S 形升降速的 S 段。
语　　法：int set_s_curve（int ch，int mode）；
int set_s_section（int ch，double accel_sec，double decel_sec）；
ch：函数所操作的轴；
mode：等于 0 时，快速运动为梯形升降速；等于 1 时，为 S 形升降速；
accel_sec：S 形升速的 S 段升速值，单位：每秒脉冲数（pps）；
decel_sec：S 形减速的 S 段减速值，单位：每秒脉冲数（pps）。
调用例子：set_s_curve（2，1）；/*把第 2 轴的升降速设置为 S 形*/
set_s_section（2，100，200）；/*升速时从初速到（初速+100）之间为 S 形，从（高速-100）到高速之间为 S 形；降速时从高速到（高速-200）之间为 S 形，从（初速+200）到初速之间为 S 形。注意：其中初速和高速由 set_profile 设置。
描　　述：对于快速运动方式，为了使升降速过程更为平稳，可以采用 S 形升降速。函数 set_s_curve 用于设置指定轴在快速运行时是否采用 S 形升降速。在 S 形升降速曲线中，S 段的升降速值由 set_s_section 来设置。
返 回 值：如果调用成功，set_s_curve 和 set_s_section 返回 0 值，否则返回-1。
系　　统：DOS，WINDOWS
参　　见：set_profile

5.2 运动指令函数

按运动类型分类，主要有三种类型：点位运动、连续运动和回原点运动；按运动方式可分为独立运动和插补运动两种；按运动速度可分为常速运动和梯形速度两种。为了描述方便，下面将运动指令分为独立运动和插补运动两部分来说明。

5.2.1 独立运动函数

所谓独立运动指各轴的运动之间没有联动关系，可以是单轴运动，也可以是多轴同时按各自的速度运动。点位运动、连续运动和回原点运动都属于独立运动。

独立运动指令的函数名格式为：X_YmoveZ

其中：

X：由 con 和 fast 替代，con 表示常速运动，fast 表示快速运动；

Y：由 p、v 和 h 替代，p 表示点位运动，v 表示连续运动，h 表示回原点运动；

move：为指令主体，表示该指令为运动指令；

Z：没有时为单轴运动，为 2 时表示两轴独立运动，为 3 时表示三轴独立运动。

例如：con_vmove 为单轴的常速连续运动函数；con_pmove2 为两轴的常速点位运动函数；fast_hmove3 为三轴的快速回原点运动指令。

对于常速运动指令，运动速度由 set_conspeed 设定；对于快速运动指令，运动速度由 set_profile 设定。

下面以点位运动、连续运动和回原点运动分别说明各运动指令的含义。

一、点位运动函数

点位运动是指被控轴以各自的速度分别移动指定的距离，在到达目标位置时自动停止。注意：在两轴或三轴的点位运动函数中，各轴同时开始运动，但不一定同时到达目标位置。在 MPC01 函数库中共提供了六个点位运动指令函数：

int con_pmove（int ch，long step）；/*一个轴以常速做点位运动*/

int fast_pmove（int ch，long step）；/*一个轴以快速做点位运动*/

int con_pmove2（int ch1，long step1，int ch2，long step2）；/*两轴以常速做点位运动*/

int fast_pmove2（int ch1，long step1，int ch2，long step2）；/*两轴以快速做点位运动*/

int con_pmove3（int ch1，long step1，int ch2，long step2，int ch3，long step3）；
/*三个轴以常速做点位运动*/

int fast_pmove3（int ch1，long step1，int ch2，long step2，int ch3，long step3）；
/*三个轴以快速作点位运动*/

其中：

ch、ch1、ch2、ch3：被控轴的轴号；

step、step1、step2、step3：表示被控轴从当前位置开始移动的距离，正数表示正方向；负数表示负方向，其单位为脉冲数。

调用例子：

con_pmove（1，-2000）；/*第一轴以其常速向负方向移动 2000 个脉冲的距离*/

fast_pmove2（2，5000，3，-1000）；/*第二轴以快速向正方向移动 5000 个脉冲的距离；第三轴以快速向负方向移动 1000 个脉冲的距离。*/

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

二、连续运动函数

连续运动是指被控轴以各自的速度按给定的方向一直运动，直到碰到限位开关或调用制动函数才会停止。在 MPC01 函数库中共提供了六个连续运动指令函数：

int con_vmove（int ch，int dir）；/*一轴以常速做连续运动*/

int fast_vmove（int ch，int dir）；/*一轴以快速连续运动*/

int con_vmove2（int ch1，int dir1，int ch2，int dir2）；/*两轴以常速连续运动*/

int fast_vmove2（int ch1，int dir1，int ch2，int dir2）；/*两轴以快速连续运动*/

int con_vmove3（int ch1，int dir1，int ch2，int dir2，int ch3，int dir3）；
/*三个轴以常速连续运动*/

int fast_vmove3（int ch1，int dir1，int ch2，int dir2，int ch3，int dir3）；
/*三个轴以快速连续运动*/

其中：

ch、ch1、ch2、ch3：被控轴的轴号；

dir、dir1、dir2、dir3：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。

调用例子：

con_vmove（1，-1）；/*第一轴以其常速向负方向连续运动*/

fast_vmove2（2，1，3，-1）；/*第二轴快速向正方向连续运动；第三轴快速向负方向连续运动。*/

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

三、回原点函数

回原点运动是指被控轴以各自的速度按给定的方向一直运动，直到碰到原点信号（参见 set_home_mode）、限位开关或调用制动函数才会停止。在 MPC01 函数库中共提供了六个回原点运动指令函数：

int con_hmove（int ch，int dir）；/*以常速返回原点*/

int fast_hmove（int ch，int dir）；/*以快速返回原点*/

int con_hmove2（int ch1，int dir1，int ch2，int dir2）；/*两轴以常速各自返回原点*/

int fast_hmove2（int ch1，int dir1，int ch2，int dir2）；/*两轴以快速各自返回原点*/

int con_hmove3（int ch1，int dir1，int ch2，int dir2，int ch3，int dir3）；

/*三个轴以常速各自返回原点*/

int fast_hmove3（int ch1，int dir1，int ch2，int dir2，int ch3，int dir3）；

/*三个轴以快速各自返回原点*/

其中：

ch、ch1、ch2、ch3：被控轴的轴号；

dir、dir1、dir2、dir3：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。

调用例子：

con_hmove（1，-1）；/*第一轴以其常速向负方向作回原点运动*/

fast_hmove2（2，1，3，-1）；/*第二轴快速向正方向作回原点运动；第三轴快速向负方向作回原点运动。*/

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

注释：要成功地实现回原点运动，运动轴上应设有常开型原点开关（接近开关或传感器），低电平或下降沿有效。

5.2.2 插补运动函数

插补运动是指两轴或三轴按照一定的算法进行联动，被控轴同时启动，并同时到达目标位置。插补运动以矢量速度运行，矢量速度分为常矢量速度和梯形矢量速度。与插补运动有关的函数有：

一、常矢量速度、梯形矢量速度等设置和读取函数

int set_vector_conspeed（int con_speed）；/*设置常矢量速度*/

int set_vector_profile（double FL，double FH，double AD）；

/*设置梯形矢量速度*/
double get_vector_conspeed（void）；/*读取矢量常速度*/
int get_vector_profile（double *vec_fl，double *vec_fh，double *vec_acc）；
/*读取矢量梯形速度*/
int set_ellipse_ratio（float ratio）；/*设置椭圆运动时的两运动轴的比例*/

函 数 名：set_vector_conspeed，get_vector_conspeed
目　　的：用 set_vector_conspeed 函数来设置常速方式下的矢量速度，这个矢量速度在两轴或三轴插补运动（包括直线、圆弧）中将会用到；用 get_vector_conspeed 函数来读取常速方式下的矢量速度。
语　　法：int set_vector_conspeed（double vec_conspeed）；
vec_conspeed：在常速插补期间的矢量速度；
double get_vector_conspeed（void）；
调用例子：set_vector_conspeed（1000）；
vec_conspeed= get_vector_conspeed（）；
描　　述：函数 set_vector_conspeed 为下列二轴或三轴插补运动函数设置矢量速度：con_line2、con_line3、arc_center、arc_final 和 helical_move 等。它不能为 fast_lin2、 fast_line3 等高速插补运动设置运动速度（它们的速度依赖于 set_vector_profile）。函数 get_vector_conspeed 返回常矢量速度。最后一次调用 set_vector_conspeed 的常矢量速度有效。
系　　统：DOS，WINDOWS
注　　释：常矢量速度应设置为相对较小一些，以免在运动过程中丢步。对于高速运动插补（如：fast_line2、fast_line3 等）来说，可用 set_vector_conspeed 来设置运动速度。
参　　见：set_vector_profile ，set_conspeed，set_profile

函 数 名：set_vector_profile，get_vector_profile
目　　的：用 set_vector_profile 来设置矢量梯形速度参数；
用 get_vector_profile 来获取矢量梯形速度参数值；
语　　法：int set_vector_profile（double vec_fl，double vec_fh，double vec_ad）；
vec_fl: 矢量低速的速度值；
vec_fh；矢量高速的速度值；
vec_ad：矢量高速的速度值
int get_vector_profile（double *vec_fc，double *vec_fh，double *vec_ad）；
*vec_fc:指向矢量低速的指针；
*vec_fh:指向矢量高速的指针；
*vec_ad：指向矢量加速度的指针。
调用例子：set_vector_profile（1000，16000，10000）；
get_vector_profile（&vec_fl，&vec_fh，&vec_ad）；
描　　述：函数 set_vector_profile 为 fast_line2，fast_line3 等函数设置矢量梯形速度。这个函数不为 con_line2，con_line3，arc_center，arc_final 及 helical_move 等函数设置运动速度。
返 回 值：如果调用成功，set_vector_profile 和 get_vector_profile 函数返回 0，在出

错的情况下，返回-1。

系　　统：DOS，WINDOWS

注　　释：对于 MPC01 控制卡来说，螺旋线插补运动没有快速运动的版本。

参　　见：set_vector_conspeed，fast_line2，fast_line3

函 数 名：set_ellipse_ratio

目　　的：用 set_ellipse_radio 来改变 X 轴和 Y 轴之间的比例，这比例仅对圆弧系列函数发生影响。

语　　法：int set_ellipse_ratio（float ratio）；

ratio：X 轴和 Y 轴之间的比例值。

调用例子：set_ellipse_radio（0.5）；

描　　述：函数 set_ellipse_radio 在圆弧系列运动（包括 arc_center、arc_final 和 helical_move）中设定 X、Y 两轴的比例，以便让其走出椭圆轨迹。一般来说，我们习惯将圆弧函数中的第一轴称为 X 轴，另一个轴为 Y 轴。在设计程序的过程中，我们假定 X 轴的比例不变（恒为 1），而全局变量‘ellipse_ratio=X/Y’，所以 set_ellipse_radio 函数仅缩放 Y 轴。

返 回 值：如果调用成功，set_ellipse_radio 函数返回 0，在出错的情况下，返回-1。

系　　统：DOS，WINDOWS

注　　释：全局变量 ‘ellipse_ratio’在缺省情况下等于 1。这就是说此时圆弧运动函数画出的是一个圆，set_ellipse_ratio 对直线插补函数没有影响。

参　　见：arc_center，arc_final，helical_move

二、线性插补函数

线性插补运动是指两个轴或三个轴以矢量速度（常矢量速度或梯形矢量速度）作线性联动，每个被控轴的运动速度为矢量速度在该轴上的分速度，各个被控轴同时启动，并同时到达目标位置。MPC01 函数库中提供四个线性插补函数：

int con_line2（int ch1，long pos1，int ch2，long pos2）；

/*两轴做常速直线运动*/

int fast_line2（int ch1，long pos1，int ch2，long pos3）；

/*两轴做快速直线运动*/

int con_line3（int ch1，long pos1，int ch2，long pos2，int ch3，long pos3）；

/*三个轴直线运动*/

int fast_line3（int ch1，long pos1，int ch2，long pos2，int ch3，long pos3）；

/*三个轴做快速直线运动*/

其中：

ch1、ch2、ch3：被控轴的轴号；

pos1、pos2、pos3：表示被控轴从当前位置开始移动的距离，正数表示正方向；负数表示负方向，其单位为脉冲数。

调用例子：

con_line2（1，-2000，3，1000）；

/*第一轴和第三轴以常矢量速度作线性插补运动，第一轴向负方向移动 2000 个脉冲的距离，同时第三轴向正向移动 1000 个脉冲的距离*/

fast_line3（2，5000，3，-1000，5，3000）；

/*第二轴、第三轴和第五轴以梯形矢量速度作线性插补运动，第二轴向正方向移动 5000 个脉冲的距离;第三轴向负方向移动 1000 个脉冲的距离；第五轴向正方向移动 3000 个脉冲的距离。*/

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

三、园弧插补函数

圆弧插补运动是指两个轴以矢量速度（常矢量速度或梯形矢量速度）沿给定圆弧作联动，每个被控轴的运动速度为矢量速度在该轴上的分速度，由于矢量速度的方向在不断改变，各被控轴的运动分速度也在不断改变。各个被控轴同时启动，并同时到达目标位置，并且两个轴的合成运动轨迹为给定的圆弧。MPC01 函数库中提供四个圆弧类插补函数：

int arc_certer（int ch1，int ch2，long cen1，long cen2，float angle）；

/*两轴以常矢量速度做园弧插补运动*/

int fast_arc_center（int ch1，int ch2，long cen1，long cen2，float angle）；

/*两轴以梯形矢量速度做园弧插补运动*/

int arc_final（int ch1，int ch2，int dir，long fx，long fy，long radius）；

/*以常矢量速度沿园弧运动到指定位置*/

int helical_move（int CHX，int CHY，int CHZ，long cen1，long cen2，float angle，long pitch）；

/*在 CHX、CHY 作圆弧插补运动的同时，CHZ 轴作直线运动*/

下面对上述函数分别详细说明。

函 数 名：arc_center，fast_arc_center

目　　的：用 arc_center 函数让两轴以常矢量速度做圆弧运动。

语　　法：int arc_center（int ch1，int ch2，long center1，long center2，float angle）；
int fast_arc_center（int ch1，int ch2，long center1，long center2，float angle）；
ch1，ch2：做圆弧运动的两个轴；
center1：轴 ch1 相对于圆弧运动起始点的圆心坐标；
center2：轴 ch2 相对于圆弧运动起始点的圆心坐标；
angle：圆弧角，单位为度。angle<0 逆时针；angle>0 顺时针。

调用例子：arc_center（1，3，-2000，0，5400）；
fast_arc_center（1，2，1000，1000，360）；

描　　述：函数 arc_center 使两个轴同时运动形成一个圆弧，其矢量速度为由 set_vector_conspeed 设置的常矢量速度。函数 fast_arc_center 使两个轴同时运动形成一个圆弧，其矢量速度为由 set_vector_profile 设置的梯形矢量速度。

返 回 值：如果调用成功，函数返回 0，在出错的情况下，返回-1。

系　　统：DOS，WINDOWS

注　　释：arc_center 可以做任意角度的圆弧运动。如果调用 set_ellipse_ratio，可以让 arc_center 走出椭圆轨迹。fast_arc_center 功能相同，速度为梯形速度。

参　　见：arc_final，set_ ellipse_ratio

函 数 名：arc_final

目　　的：用 arc_final 函数来使两个轴在平面上做圆弧运动（圆弧角不超过 180 度）

语　　法：int arc_final（int ch1，int ch2，int dir，long final1，long final2，long radius）；
ch1，ch2；做圆弧运动的两个轴；
dir：圆弧运动方向，-1 对应于逆时针，1 对应于顺时针；
final1：轴 ch1 的终点位置，相对于圆弧运动起始点；
final2：轴 ch2 的终点位置，相对于圆弧运动起始点；
radius：圆弧半径。

调用例子：arc_final（1，2，-1，-2000，1000，3000）；

描　　述：函数 arc_final 使两个指定轴以常矢量速度从起始点圆弧按指定的方向运动到指定的终点位置。在这个函数中，ch1 和 ch2 是指定参与圆弧运动中涉及到的两个轴，参数 ‘dir’指明运动方向（CW 或 CCW），圆弧终点位置由参数 final1、final2 指定，参数 radius 是圆弧的半径。

返 回 值：如果调用成功，arc_final 函数返回 0，在出错的情况下返回-1。

系　　统：DOS，WINDOWS

注　　释：arc_final 的圆弧角不能大于 180 度，只能做圆弧角小于 180 度的圆弧运动。如果调用 set_ellipse_ratio，可以让 arc_final 走出椭圆轨迹。

参　　见：arc_center

函 数 名：helical_move

目　　的：用 helical_move 函数让给定的两轴以常矢量速度做圆弧运动，第三轴以给定的比例在垂直方向上做线性运动。

语　　法：int helical_ove（int CHX，int CHY，int CHZ，long cen1，long cen2，float angle，long pitch）；
CHX、CHY：做圆弧运动的两个轴；
CHZ：在垂直方向上按比例做线性运动的轴；
cen1：轴 CHX 相对于圆弧运动起始点的圆心坐标；
cen2：轴 CHY 相对于圆弧运动起始点的圆心坐标；
angle：圆弧的角度，单位为度正值为顺时针，负值为逆时针
pitch：圆弧每走 360 度（一周）第三轴所走的位移。

调用例子：helical_move（1，2，3，20000，-20000，-360，30000）；

描　　述：函数 helical_move 使指定的两个轴以常矢量速度沿圆弧运动，而第三轴按参数 ‘pitch’给定的比例在垂直方向上做线性运动。常矢量速度由 set_vector_conspeed 设置。在这个函数中，CHX 和 CHY 是圆弧插补运动中涉及到的两个轴，圆心位置 cen1 和 cen2 是相对于圆弧运动起始点的。参数 long cen1，long cen2：轴 CHX 和 CHY 相对于圆弧运动起始点的圆心坐标；‘angle’ 是沿圆弧走的角度。如果 angle>0，则圆弧运动方向为顺时针，反之为逆时针，参数 ‘pitch’定义垂直的运动距离的比例因子，即圆弧每运动一周时第三轴所运动的距离。

系　　统：DOS，WINDOWS

返 回 值：如果调用成功，helical_move 函数返回 0，在出错的情况下，返回-1。

注　　释：helical_move 可以以常矢量速度做任意角度的螺旋运动，如果调用 set_ellipse_ratio，可以让 helical_move 走出椭圆螺旋轨迹。

参　　见：arc_center，set_ellipse_ratio

5.3 制动函数

在运动过程中，如果需要暂停或中止某个轴或某几个轴的运动，可以调用制动函数来完成。在 MPC01 运动函数库中提供了六个制动函数：

void sudden_stop（int ch）；*/立即制动一个运动轴*/
void sudden_stop2（int ch1，int ch2）；*/立即制动两个运动轴*/
void sudden_stop3（int ch1，int ch2，int ch3）；*/立即制动三个运动轴*/
void decel_stop（int ch）；/*光滑制动一个运动轴*/
void decel_stop2（int ch1，int ch2）；/*光滑制动两个运动轴*/
void decel_stop3（int ch1，int ch2，int ch3）；/*光滑制动三个运动轴*/
其中：
　　ch、ch1、ch2、ch3：被控轴的轴号；
调用例子：
　　decel_stop（2）；/*光滑制动第二轴*/
　　sudden _stop2（1，4）；/*立即制动第一、四轴*/
　　decel_stop3（1，2，3）；/*光滑制动第一、二、三轴*/
返回值：无

说明：制动函数对所有类型的运动都有效。decel_stop 类型的制动函数一般用于梯形速度或 S 形速度的运动方式（fast_YmoveZ），它可以使被控轴的速度先从高速降至低速（由 set_proflile 设定），然后停止运动。一般在运动过程需要暂停时应调用 decel 类型制动函数，以便能够光滑地中止快速运动（如：fast_hmove、fast_vmove、fast_pmove2 等），以免发生过冲现象。sudden_stop 类型制动函数使被控轴立即中止运动，这个函数执行后，控制卡立即停止向电机驱动器发送脉冲，使之停止运动。该函数通常在紧急停车时调用。对于常速运动方式（con_YmoveZ），这两类制动函数效果一样。

5.4 位置和状态查询函数

在运动过程中，如果需要查询某个轴的运动位置或状态，可以调用位置或状态查询函数。

5.4.1 位置查询函数

int set_getpos_mode(int ch，int mode) /*设置 get_encoder()函数返回值的来源*/
int set_encoder_mode(int ch，int mode，int multip，int count_unit)
　　/*设置一个轴的编码器反馈信号模式和计数单位等*/

int get_abs_pos（int ch，long*pos）；/*返回一个轴的绝对位置值*/
int get_rel_pos（int ch，long*pos）；/*返回一个轴的相对位置值*/
int get_encoder（int ch，long*pos）；/*返回一个轴的实际位置值*/

函 数 名：set_getpos_mode
目　　的：用于设置调用 get_encoder()函数获取的位置值的来源。
语　　法：int set_getpos_mode（int ch，int mode）；
ch：所设置的控制轴；
mode：调用 get_encoder()函数获取的位置值的来源（1 为编码器反馈信号，0 为输出脉冲数）；
调用例子：set_ getpos _mode（1，1）；/*将第 1 轴设定为编码器反馈信号*/
描　　述：缺省情况下调用 get_encoder()函数获取的位置值为实际输出脉冲数，若要让调用 get_encoder 函数获取的位置值为编码器反馈信号，则应在调用 init_board()函数之后调用该函数进行设置，否则调用 get_encoder 函数获取的位置值将不是读取的编码器位置。
返 回 值：如果设置成功，则 set_ getpos _mode 返回值为 0，否则返回-1。
系　　统：DOS，WINDOWS
参　　见：get_encoder

函 数 名：set_encoder_mode
目　　的：用于设置每个轴的编码器反馈信号模式和计数单位等。
语　　法：int set_encoder_mode（int ch，int mode，int multip，int count_unit）；
ch：所设置的控制轴；
mode：编码器反馈信号模式（1 为 A/B 相 90 度相位差方式，0 为增减脉冲方式：A 为增脉冲，B 为减脉冲）；
multip：A/B 相 90 度相位差方式时的倍频：1、2、4。
count_unit：对反馈脉冲的计数单位：1~16。
调用例子：set_encoder_mode（1，0，0，2）；/*将第 1 轴的编码器反馈信号设置为增减脉冲方式，且每 2 个脉冲计数一次。*/
描　　述：在缺省情况下，init_board 函数将所有轴设置为 A/B 相 90 度相位差的编码器反馈信号，4 倍频，每 1 个脉冲计数一次，那么应在 init_board 函数后调用 set_encoder_mode 重新设置所要求的模式。注意：控制卡的编码器模式设置应与所连接的编码器的反馈信号模式一致，否则读取的编码器位置（get_encoder）将不正常。
返 回 值：如果设置成功，则 set_encoder_mode 返回值为 0，否则返回-1。
系　　统：DOS，WINDOWS

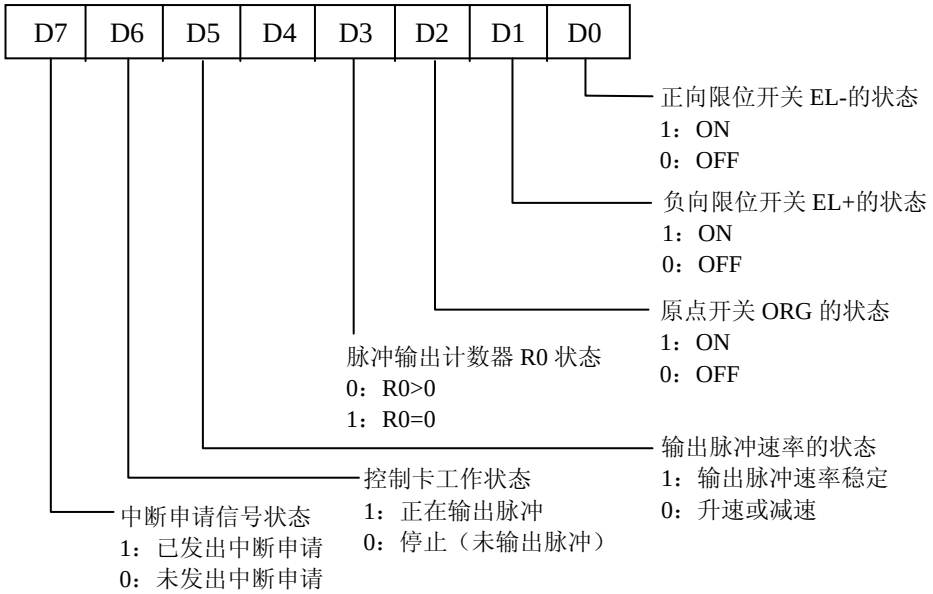
函 数 名：get_abs_pos，get_rel_pos，get_encoder
目　　的：用 get_abs_pos 读取一个相对于初始位置或原点位置的绝对位置。
用 get_rel_pos 读取一个相对于当前运动起始点的相对位置值。
用 get_encoder 读取一个相对于初始位置或原点位置的编码器反馈的实际位置值。
语　　法：int get_abs_pos（int ch，long*abs_pos）；
int get_rel_pos（int ch，long*rel_pos）；
int get_encoder（int ch，long*en_pos）；
ch：读取位置的轴号；
abs_pos：一个指向绝对位置的长整型指针；

rel_pos：一个指向相对位置的长整型指针；
en_pos：一个指向实际位置的长整型指针；
调用例子：temp=get_abs_pos（1，&abs_pos）；
temp=get_rel_pos（1，&rel_pos）；
描　　述：函数 get_abs_pos 获取指定轴的当前绝对位置，如果执行过回原点运动，那么这个绝对位置是相对于原点位置的；如果没有执行过回原点运动，那么这个绝对位置是相对于开机时的位置。函数 get_rel_pos 获取对应于当前运动起始点的相对位置值，如果指定轴当前没有运动，那么该轴的相对位置为 0。由于这两个函数读取的位置值是由控制卡输出脉冲的数量决定的，所以在丢步或过冲等情况下，不能反映实际的位置值。get_encoder 读取光电盘反馈的实际位置。
返 回 值：如果调用成功，get_abs_pos、get_rel_pos 和 get_encoder 返回 0 值，在出错情况下返回-1。
系　　统：DOS，WINDOWS

5.4.2 状态查询函数

int check_status（int ch）；/*检查一个轴的状态值*/
int check_done（int ch）；/*检测一个轴的运动是否完成*/
int check_board（void）；/*检查控制卡是否已经插入计算机的扩展槽*/
int check_limit（int ch）；/*检测一个轴指定的限位开关是否闭合*/
int check_home(int ch)；/*检查一个轴是否已经到达原点开关位置*/

函 数 名：check_status
目　　的：用 check_status 函数读取并返回一个轴的状态值。
语　　法：int check_status（int ch）；
ch：所读取状态的轴号。
调用例子：ch_status=check_status（2）；
描　　述：函数 check_status 读取指定轴的状态。MPC01 控制卡每个轴都有 1 个单字节的状态值，用于查询轴的工作状态。该字节中每位（bit）的含义如下图所示。
返 回 值：如果调用成功，check_status 返回指定轴的状态值，在出错时返回-1。
系　　统：DOS，WINDOWS
参　　见：wait_done



函 数 名：check_done

目　　的：用 check_done 函数来检查指定轴的运动是否已经完毕。

语　　法：int check_done（int ch）；
ch：所检查的轴号。

描　　述：函数 check_done 检查指定轴是在运动中还是在静止状态。

返 回 值：如果指定轴正在运动状态，check_done 返回-1，如果指定轴正在静止状态，check_done 返回 0。

系　　统：DOS，WINDOWS

参　　见：wait_done

函 数 名：check_board

目　　的：用 check_board 函数来检查 PC 机上是否有 MPC01 控制卡以及各轴的内部寄存器是否正常。

语　　法：int check_board（void）；

描　　述：在控制卡各参数设置和初始化后，用函数 check_board 检查 MPC01 卡是否存在，并检测各轴的内部寄存器是否能正常工作。一般利用此函数来完成控制系统的自检工作。

返 回 值：如果控制卡检查通过，check_board 返回 0，否则 check_board 返回有问题的轴号。

系　　统：DOS，WINDOWS

参　　见：

函 数 名：check_limit

目　　的：用 check_limit 函数来检查一个轴是否已经到达限位开关位置。

语　　法：int check_limit（int ch）；

ch：所检查的轴号；
调用例子：status=check_limit（1）；
描　　述：函数 check_limit 用于检测指定轴的限位开关，并返回指定轴的位置是否已经到达限位开关位置及到达哪一个方向上的限位开关位置。
返 回 值：如果 check_limit 返回 1 表示到达正向限位开关位置，返回-1 表示到达负向限位开关位置，返回 0 则表示未到达限位开关位置，若调用出错则返回-2。
系　　统：DOS，WINDOWS

函 数 名：check_home
目　　的：用 check_home 函数来检查一个轴是否已经到达原点开关位置。
语　　法：int check_home（int ch）；
ch：所检查的轴号；
调用例子：status=check_home（1）；
描　　述：函数 check_home 用于检测指定轴的原点开关，并返回指定轴的位置是否已经到达原点开关位置。
返 回 值：如果 check_home 返回 1 表示到达原点开关位置，返回 0 则表示未到达限位开关位置，若调用出错则返回-1。
系　　统：DOS，WINDOWS

5.5 中断管理和其它函数

处理中断通常是一项极具挑战性的工作，为了简化这一工作，在 MPC01 函数库中提供了一些相关的函数来处理中断。另外，还提供了一些诸如等待轴运动结束、间隙补偿等辅助函数。

中断函数（参见 4.1.5 节）

```
int enable_board_irq（）；/*使控制卡可以向 PC 机申请中断*/
int disable_board_irq（）；/*使控制卡不能向 PC 机申请中断*/
int set_board_isr（int（*my_isr）（））；
        /*让一个常规函数执行控制卡的中断服务*/
int enable_axis_intr（int ch）；/*使一个轴可以产生中断*/
int disable_axis_intr（int ch）；/*使一个轴不能产生中断*/
int enable_timer_irq（）；/*使 DOS timer 可以中断 PC 机*/
int disable_timer_irq（）；/*使 DOS timer 不能中断 PC 机*/
int set_timer_isr（int（*my_isr）（））；
  /*让一个常规函数执行 DOS timer 的中断服务*/
```

函 数 名：enable_board_irq，disable_board_irq
目　　的：用 enable_board_irq 使控制卡能向 PC 申请中断服务；
用 disable_board_irq 使控制卡不能向 PC 申请中断服务。
语　　法：int enale_board_irq（）；
int disable_board_irq（）；

调用例子：enable_board_irq（）；
disable_board_irq（）；
描　　述：函数 enable_board_irq 让控制卡可以向 PC 申请中断服务，而调用 disable_board_irq 函数后，则不允许控制卡可以向 PC 申请中断服务。一般来说，在调用 enable_board_irq 前应先调用 set_board_irq 和 set_board_isr 两个函数，如果已经调用了 enable_board_irq，在您退出应用程序前，一定要调用 disable_board_irq，否则会造成死机。
返 回 值：如果调用成功，enable_board_irq（disable_board_irq）将返回 0，否则返回-1。
系　　统：DOS
注　　释：enable_board_irq 和 disable_board_irq 是一对中断管理函数，在应用程序中它们应成对使用。
参　　见：set_board_irq，get_board_irq，set_board_isr，enable_axis_intr

函 数 名：enable_axis_intr，disable_axis_intr
目　　的：用 enable_axis_intr 使控制器的一个轴可以发出中断申请信号；
用 disable_axis_intr 使控制器的一个轴不能发出中断申请信号；
语　　法：int enable_axis_intr（int ch）；
int disable_axis_intr（int ch）；
ch：被控轴的轴号；
调用例子：enable_axis_intr（2）；
disable_axis_intr（2）
描　　述：函数 enable_axis_intr 使指定的控制器轴具有中断能力，可以发出中断申请信号，而 disable_axis_intr 使指定的控制器轴不具有中断能力，不能发出中断信号，如果一个轴具有中断能力，当这个轴的运动完毕，它将会向 PC 发出中断申请信号。
返 回 值：如果调用成功 enable_axis_intr（disable_axis_intr）返回 0，否则返回-1。
系　　统：DOS
注　　释：在调用 enable_axis_intr 前一般要先用函数 set_board_irq，set_board_isr 和 enable_axis_intr 否则可能出现无法预料的结果。
参　　见：set_board_irq，get_board_irq，set_board_isr

函 数 名：set_board_isr
语　　法：int set_board_isr，（int（*my_isr）（））；
（int（*my_isr（（））；指向一个常规整形函数的函数指针；
调用例子：set_board_isr（my_isr）；
描　　述：函数 set_board_isr，可以让一个常规函数充当中断服务程序的角色处理中断，如果 set_board_isr，这个函数和 enable_boable_irq 均已调用，并且控制卡的某轴已具备中断能力（调 enable_axis_intr），那么每当这个轴的运动完毕时，这个常规函数就会被调用。
返 回 值：如果钩往一个函数成功，set_board_isr 返回 0，否则返回-1。
系　　统：DOS
注　　释：单独调用 set_board_isr 并无任何意义，对程序也不会有任何影响，只有

和 set_board_isr，enable_board_irq，enable_axis_intr 在一起调用，钩住的那个常规函数才可能发生作用。

参　　见：set_board_isr，enable_board_irq，enable_axis_intr

函 数 名：enable_timer_irq，disable_timer_irq

目　　的：用 enable_timer_irq，让 DOS timer 可以向 PC 申请中断服务
用 disable_timer_irq 不能向 PC 申请中断服务；

语　　法：int enable_timer_irq（）
int disable_timer_irq（）；

调用例子：enable_timer_irq（）；
disable_timer_irq（）；

描　　述：函数 enable_timer_irq 让 DOS timer 可以向 PC 申请中断服务，而调用 disable_timer_irq 前您只需调用函数 set_timer_isr，并写一个常规函数“my_isr“（也可以是其它函数名），您就可以处理中断了。如果您已经调用了 enable_timer_irq 函数，那么您在退出应用程序前，一定要调用 disable_timer_irq。

系　　统：DOS

参　　见：set_timer_isr，enable_timer_irq，disable_timer_irq

函 数 名：set_timer_isr

目　　的：用 set_timer_isr 函数可以使一个常规函数钩住 DOS　timer 中断，以便使这个常规函数可以处理中断，充当中断服务程序的角色。

语　　法：int set_timer_isr（int（*my_isr）（）；
（int（*my_isr））指向一个常规的整形函数的函数指针；

调用例子：set_timer_isr（*my_isr）

描　　述：函数 set_timer_isr 可以让一个常规函数充当中断服务程序的角色，处理 DOS 定时中断，如果它和 enable_board_irq 均已正确调用，那么在每个 DOS 定时周期（55ms）这个常规函数就会被调用一次。

返 回 值：如果钩住一个函数成功，set_timer_isr 返回 0，否则返回-1。

系　　统：DOS

注　　释：单独调用 set_timer_isr 并无任何意义，对程序也不会有任何影响，只有和 enable_timer_irq 一起调用，钩住那个常规函数才可能发生作用。注意：在这个钩住的常规函数中，不应调用与中断处理相冲突的 C 语言库函数，在 DOS 中断服务程序中，　一般不要调用另一个中断事务函数（如 printf 等）。

参　　见：enable_timer_irq，disable_timer_irq

其它函数

int reset_pos（int ch）；/*当前位置值复位至零*/
int wait_done（int ch）　/*等待一个轴的运动完结*/
int wait_done2（int ch1，int ch2）　/*等待两个轴的运动完结*/
int wait_done3（int ch1，int ch2，int ch3）　/*等待三个轴的运动完结*/
int set_backlash（int ch，int backlash）/*设置由于机构换向形成间隙的补偿值*/

int start_backlash（int ch）；/*开始间隙补偿*/
itn end_backlash（int ch）；/*终止间隙补偿（end backlash compensation）*/

函 数 名：reset_pos
语　　法：int reset_pos（int ch）；
　　ch：被复位轴的轴号；
调用例子：int reset_pos（1）；
描　　述：函数 reset_pos 将指定轴的绝对位置和相对位置复位至 0，通常在轴的原点找到时调用，调用这个函数后，当前位置值变为 0，这以后，所有的绝对位置值均是相对于这一点的。
返 回 值：如果调用成功，reset_pos 返回 0，否则返回-1。
系　　统：DOS，WINDOWS
注　　释：一般来说，这个函数应在成功地执行 con_hmove 或 fast_hmove 后调用。
参　　见：get_abs_pos，get_rel_pos

函 数 名：wait_done，waite_done2，waite_done3
目　　的：用 wait_done 函数等待一个轴的运动结束；
　　用 waite_done2 函数让 PC 等待，直到两个轴的运动结束；
　　用 waite_done3 函数让 PC 等待，直到三个轴的运动结束；
语　　法：int wait_done（int ch）；
　　int wait_done2（int ch int ch2）；
　　int wait_done3（int ch1，int ch2，int ch3）；
　　ch、ch1、ch2、ch3：被控轴的轴号。
调用例子：wait_done（1）；
　　wait_done2（1，3）；
描　　述：函数 wait_done 让主 PC 一直等待直到前一个运动结束，这通常是为了防止上一个运动还没有执行完，又执行下一条运动指令，函数 wait_done2 让主 PC 等待两个指定轴的运动结束，只有当两个指定轴的运动都完毕时，它才返回。
返 回 值：如果运动结束，wait_done（waite-done2）返回 0，如果在等待期间，按下‘S’键，返回-1，按下‘D’键，返回-2。
系　　统：DOS
参　　见：check_done

函 数 名：set_backlash，start_backlash，end_backlash
目　　的：用 set_backlash 设置补偿由于机构换向形成间隙的补偿值。
　　用 start_backlash 开始补偿由于机构换向间隙而导致的位置误差。
　　用 end_backlash 停止补偿由于机构换向间隙而导致的位置误差。
语　　法：int set_backlash（int ch，int backlash）；
　　int start_backlash（int ch）；
　　int end_backlash（int ch）
　　ch：轴号；
　　backlash：由于机构换向形成的间隙值，单位为脉冲数；

调用例子：set_backlash（1，12）；
start_backlash（1）；
end_backlash（1）

描　　述：函数 set_backlash 设置一个补偿，以便消除由于机构换向形成的位置误差。函数 start_backlash（CH1）调用后，开始补偿。补偿值是通过调用 set_backlash 设置的。

返 回 值：如果设置成功，set_backlash、set_backlash 和 end_backlash 返回 0，否则返回-1。

系　　统：DOS，WINDOWS

注　　释：set_backlash 函数仅是设置补偿值，真正的补偿直到调用函数 start_backlash 才起作用。函数 set_backlash 应在调用 start_backlash 前调用，否则系统采用缺省补偿值（缺省值为 10 步）。

参　　见：

6 新增功能及函数

6.1 MPC01(3.2 版)软件概述

MPC01（3.2 版）是原 MPC01（3.0 版）的最新升级版本。该版本主要对 Windows 环境下的动态链接库代码部分做了改进与升级，DOS 环境下的代码不变。

以下内容均是针对 Windows 环境下的动态链接库而言。

MPC01（3.2 版）采用虚拟设备驱动程序（Vxd）技术开发，充分利用 MPC01 板卡中断及 Windows 底层控制编程，提高了 Windows 环境下软件运行效率及控制的实时性，改善了在多任务环境下运行状态，使得软件不易受多任务影响而产生运动控制的不连续性，并可在混合运动指令连续执行期间调用相应函数获得当前位置，通过编程进行实时的图形显示。

新版软件保持了与 3.0 版本的兼容性，并扩充了函数库，使得运动指令可以两种方式执行：（1）立即方式，即不等上一条指令执行完毕则下一条指令立即发出；（2）批处理方式，即只有上一条运动指令控制的所有轴都已经进给完毕，下一条指令才能发出。

6.2 新增功能说明

6.2.1 运动指令执行方式

运动指令可以两种方式运行，批处理方式和立即方式。

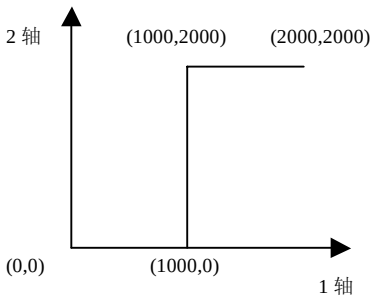
（1） 批处理方式主要在多条不同的运动指令连续执行时使用。在这种方式下只有上一条运动指令控制的所有轴全部运动完毕后才开始下一条运动指令的执行。

（2） 立即方式主要在单指令实时控制不同的轴运动时使用。该方式下不等上一条运动指令控制的所有轴运动完毕即开始下一条运动指令的执行。若新发出指令控制的轴只要未处于运动状态则立即开始按新运动指令运动，否则新发出指令不予执行并返回一个错误。这种方式若用在多条不同的运动指令连续执行时使用可能造成某些指令无法执行，除非开发人员通过检查运动状态或错误代码加以避免。

以下代码示例说明同一批运动指令按不同方式执行后的区别：（假设 1、2 两轴速度设置相同）

（1） 批处理方式

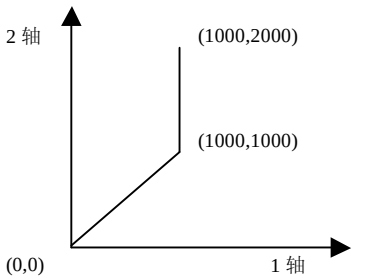
……
set_batch_flag(1);
con_pmove(1,1000);
con_pmove(2,2000);
con_pmove(1,1000);
……
运行后运动轨迹如下：



批处理方式

（2） 立即方式

……
set_batch_flag(0);
con_pmove(1,1000);
con_pmove(2,2000);
con_pmove(1,1000);
……
运行后运动轨迹如下：（其中第三条指令执行不到，因为第三条指令发出时第一条指令控制的 1 轴正在运动。）



立即方式

注意：

- 系统缺省处于立即执行方式。
- 调用 set_batch_flag()函数修改执行方式时应在所有运动进行完毕后执行。修改后系统将一直处于该方式，直到重新修改或退出程序为止。
- 圆弧运动指令不受方式改变的影响。

6.2.2 获取错误代码

该版本在保留各 DLL 函数返回值的同时，还提供了单独的获取错误代码函数，用于连续执行一批指令时通过检查错误代码来判断是否中间过程发生了错误。

通过 get_last_err()函数可获取最近一次错误代码。通过 get_err()函数可获取最近十次产生错误的错误代码。各错误代码含义见 6.6 节。

6.3 运动控制系统的开发

6.3.1 MPC01（3.2 版）软件的使用要求

为保证 V3.2 版本软件正常运行，软硬件方面应满足如下要求：

（1） 软件可以在 Windows 9X（95、97、98）、WindowsNT 或 Windows2000 操作系统中运行，需要安装不同的驱动程序。

（2） 板卡必须按照操作手册第 6 页至第 7 页内容正确设置基地址和中断号，并正确插入到计算机中。

（3） 程序中必须按照板卡上设置的中断号和基地址调用相应的库函数进行设置。

6.3.2 MPC01（3.2 版）软件的安装

DOS 环境下软件的安装仍按操作手册第 8 页内容进行。

以下为 Windows 环境下软件的安装方法，找出随卡附带的安装光盘并按如下步骤进行安装：

（1） 将 MPC01 卡软件安装光盘放入 CDROM 驱动器。

（2） 打开\Dmc300\Dmc300(3.1)\Setup\Disk1 文件夹，运行 Setup.exe 启动安装程序。

（3） 按界面提示完成安装过程。

6.3.3 开发 Windows 环境下的运动控制系统

6.3.3.1 用 Visual Basic 开发控制程序

（一） 开发环境

用户可以使用 VB5.0 或更高版本，来进行 Windows 环境下运动控制系统开发。

（二） 动态链接库函数调用方法

在 VB 中调用动态链接库 DLL 中函数应包括两部分工作：

- 函数声明

每一个动态链接库 DLL 中的函数在 VB 中的声明已经包含在 dmc300.bas 文件中了，该文件可在 MPC01 卡软件安装盘\Dmc300\V31\DLL\VB 文件夹下找到，用户只需要将该文件添加进 VB 工程中即可。

- 函数调用

若调用函数的返回值为空或不需要返回值，则按如下方法调用：

```
con_pmove   1,2000
或
call con_pmove(1,2000)
```

若要得到函数的返回值，则按如下方法调用：

```
dim rtn as long
rtn=con_pmove(1,2000)
```

注意：传递的参数数据类型及接收返回值的变量类型应与函数声明的数据类型一致，并且建议所有整型（integer）和长整型（long）参数及返回值均统一采用长整型（long）数据类型，否则将可能产生无法预料的结果。

（三）演示示例程序的使用

MPC01 卡软件安装盘\Dmc300\V31\Demo\VBDemo\文件夹下有两个在 VB6.0 下开发的运动控制系统演示示例程序，\Dmc300\ V31\Demo\VBDemo\Demo1\文件夹下为一个简单的示例程序，\Dmc300\ V31\Demo\VBDemo\Demo2\文件夹下为一个比较复杂带有图形显示的示例程序。用户可按照如下步骤编译并运行该示例，在熟悉了相应编程方法后，用户可根据需要开发自己的运动控制系统。

（1） 按照 MPC01（3.2 版）软件的安装步骤进行正确安装。

（2） 在硬盘上建立一个文件夹。

（3） 将 MPC01 卡软件安装盘\Dmc300\ V31\Demo\VBDemo\文件夹下 Demo1 文件夹中所有文件或 Demo2 文件夹中所有文件拷贝到硬盘上所建文件夹中，并去掉所有文件的只读属性。

（4） 启动 VB6.0 集成环境，并打开工程。

（5） 确保板卡已经正确设置并插入到计算机中。

（6） 按照板卡设置的中断号修改设置中断号函数 set_board_irq()调用参数。
（7） 编译该工程生成 EXE 文件。
（8） 运行生成的 EXE 文件。

6.3.3.2 用 Visual C++开发控制程序

（一）开发环境

用户可以使用 VC5.0 或更高版本，来进行 Windows 环境下运动控制系统开发。

（二）动态链接库函数调用方法

在 VC 中调用动态链接库 DLL 中函数有两种方法：

- 隐式调用

隐式调用需要如下文件：
（1） DLL 函数声明头文件 dmc300.h；
（2） 编译连接时用的导入库文件 dmc.lib；
（3） 动态链接库文件 dmc.dll；
（4） 虚拟设备驱动程序 dmc.vxd。

以上文件中的（1）（2）可在 MPC01 卡软件安装盘\Dmc300\ V31\DLL\VC 文件夹下找到。（3）（4）则已经由安装程序安装到 C:\WINDOWS\SYSTEM 文件夹下（假定 Windows 安装在 C:\WINDOWS 文件夹下）。

建立工程之后，在 VC 集成环境中点击“/project/settings…”菜单弹出“project settings”对话框。选“Link”选项卡，在“object/library modules”栏内输入导入库文件名 dmc.lib，单击“OK”按钮。在调用 DLL 函数的源代码文件开始处包含 dmc300.h 头文件。之后则可以按照调用内部函数一样调用 DLL 函数。

- 显式调用

显式调用只需要如下文件：
（1） 动态链接库文件 dmc.dll；
（2） 虚拟设备驱动程序 dmc.vxd。

以上文件已经由安装程序安装到 C:\WINDOWS\SYSTEM 文件夹下（假定 Windows 安装在 C:\WINDOWS 文件夹下）。

显式调用方法需要调用 Windows API 函数加载和释放动态链接库。方法如下：
（1） 调用 Windows API 函数 LoadLibrary()动态加载 DLL；
（2） 调用 Windows API 函数 GetProcAddress()取得将要调用的 DLL 中函数的指针；
（3） 用函数指针调用 DLL 中函数完成相应功能；

（4） 在程序结束时或不再使用 DLL 中函数时，调用 Windows API 函数 FreeLibrary()释放动态链接库。

该方法比较烦琐。MPC01（V3.2）软件中已经将常用的 Dmc.dll 中 DLL 函数封装成类 CLoadDll，并提供该类的源代码。该类含有与运动指令库函数名及参数相同的成员函数。源代码可在 MPC01 卡软件安装盘\Dmc300\ V31\Dll\Vc\文件夹下找到，文件名为 LoadDll.cpp 和 LoadDll.h。开发人员可将其添加进工程，在程序适当地方添加该类的对象，通过对应成员函数来调用 DLL 中的函数。具体可参见演示演示示例：\Dmc300\ V31\Demo\VCDemo\Demo2。

以上在两种方法均为 VC 中调用动态链接库函数的标准方法，若要获得更具体的调用方法和帮助，请参考微软 Visual Stutio 开发文档 MSDN 相应内容。

（三）演示示例程序的使用

MPC01 卡软件安装盘\Dmc300\ V31\Demo\VCDemo\文件夹下有两个在 VC6.0 下开发的运动控制系统演示示例程序。\Dmc300\ V31\Demo\VCDemo\Demo1 为隐式调用示例；\Dmc300\ V31\Demo\VCDemo\Demo2 为显式调用示例。用户可按照如下步骤编译并运行示例，在熟悉了相应编程方法后，用户可根据需要开发自己的运动控制系统。

（1） 按照 MPC01（V3.2）软件的安装步骤进行正确安装。
（2） 在硬盘上建立一个文件夹。
（3） 将 MPC01 卡软件安装盘\Dmc300\ V31\Demo\VCDemo\文件夹下 Demo1 文件夹中所有文件或 Demo2 文件夹中所有文件拷贝到硬盘上所建文件夹中，并去掉所有文件的只读属性。
（4） 启动 VC6.0 集成环境，并打开工程 demo1.dsw 或 demo2.dsw。
（5） 确保板卡已经正确设置并插入到计算机中。
（6） 按照板卡设置的中断号修改设置中断号函数 set_board_irq()调用参数。
（7） 编译连接该工程生成 EXE 文件。
（8） 运行生成的 EXE 文件。

6.4 扩展库函数描述

6.4.1 批处理状态设置函数

函 数 名：set_batch_flag

目　　的：用于设置当前批处理标志。运动指令可以两种方式执行，即批处理方式和立即方式。其区别和作用参见运动指令执行方式一节。

语　　法：int set_batch_flag(int flag);
调用例子：set_batch_flag (0)　　/*设为立即执行方式*/
　　set_batch_flag (1)　　/*设为批处理执行方式*/
返 回 值：成功返回 0，失败返回-1
系　　统：WINDOWS
参　　见：

函 数 名：check_batch_done
目　　的：用 check_batch_done 函数来在批处理指令执行过程中检查批处理指令是否执行完毕。
语　　法：int check_batch_done（）;
描　　述：用 check_batch_done 函数来在批处理指令执行过程中检查批处理指令是否执行完毕。
返 回 值：若批处理指令执行完毕返回 0，否则返回 1。
系　　统：WINDOWS
参　　见：

6.4.2 错误代码函数

函 数 名：get_last_err
目　　的：用于获取之前指令执行过程中产生的最近一次错误的错误代码。
语　　法：int get_last_err();
调用例子：err=get_last_err();
返 回 值：最近一次错误的错误代码，0 表示没有错误。
系　　统：WINDOWS
参　　见：get_err()，reset_err()

函 数 名：get_err
目　　的：用于查询获取之前指令执行过程中产生的最近十次错误的错误代码。
语　　法：int get_err(int Index);
调用例子：err=get_err(2);　/*获取最近第 2 次产生的错误代码*/
返 回 值：最近第 Index 次错误的错误代码，0 表示没有错误；-1 表示 Index 参数超出了 1 到 10 的范围。
系　　统：WINDOWS
参　　见：get_last_err()，reset_err()

函 数 名：reset_err
目　　的：用于清除所有错误的错误代码。调用该函数之后，再调用 get_last_err()或 get_err()函数均将返回 0，即没有错误。

语　　法：int reset_err();
调用例子：err=reset_err();
返 回 值：成功返回 0，失败返回-1
系　　统：WINDOWS
参　　见：get_last_err()，get_err()

6.4.3 其它新增函数

函 数 名：get_cur_dir
目　　的：用于获取轴的当前运动方向。
语　　法：int get_cur_dir（int ch）；
　　ch：所要查询的轴；
调用例子：get_cur_dir（1）；/*获取第 1 轴的当前运动方向*/
返 回 值：如果函数调用失败，则返回值为 0；否则若返回-1 表示负向，返回 1 则表示正向。
系　　统：WINDOWS
参　　见：

函 数 名：set_abs_pos
目　　的：用于设置轴的运动起始绝对位置。
语　　法：int set_abs_pos（int ch，long pos）；
　　ch：所要设置的轴；
　　pos：所要设置的该轴的起始绝对位置；
调用例子：set_abs_pos（1，1000）；/*将第 1 轴的当前位置设置为 1000*/
描　　述：调用该函数可将当前绝对位置设置为某一个值，但从上一个位置到该位置之间不会产生轴的实际运动。
返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。
系　　统：WINDOWS
参　　见：

函 数 名：set_dir
目　　的：用于设置轴的运动方向。
语　　法：int set_dir（int ch，int dir）；
　　ch：所要设置的轴；
　　dir：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。；
调用例子：set_dir（1，-1）；/*将第 1 轴的运动方向设置成为负方向*/
描　　述：调用该函数可在新的运动指令发出前设定某轴的运动方向。
返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。
系　　统：WINDOWS

参　　见：

函 数 名：check_SD
目　　的：用 check_SD 函数来检查指定轴的外部减速信号。
语　　法：int check_SD（int ch）；
　　ch：所检查的轴号。
描　　述：函数 check_SD 检查指定轴的外部减速信号。
返 回 值：如果指定轴的外部减速信号为正向有效，check_SD 返回 1，如果指定轴的外部减速信号为负向有效，check_SD 返回-1，若没有外部减速信号，则 check_SD 返回 0。
系　　统：DOS，WINDOWS
参　　见：

函 数 名：check_alarm
目　　的：用 check_alarm 函数来检查指定轴的外部报警信号。
语　　法：int check_alarm（int ch）；
　　ch：所检查的轴号。
描　　述：函数 check_alarm 检查指定轴的外部报警信号。
返 回 值：如果指定轴的外部报警信号有效，check_alarm 返回 1，若没有外部报警信号，则 check_alarm 返回 0。
系　　统：DOS，WINDOWS
参　　见：

函 数 名：delay_time
目　　的：用 delay_time 函数来在批处理过程中使运动指令之间产生延时。
语　　法：int delay（long time）；
　　time：延时的时间（单位：毫秒）。
描　　述：用 delay_time 函数来在批处理过程中使运动指令之间产生延时，该指令应在批处理执行方式下使用。
返 回 值：正确调用返回 0，否则返回-1。
系　　统：WINDOWS
参　　见：

函 数 名：check_delay_status
目　　的：用 check_delay_status 函数来检查延时状态。
语　　法：int check_delay_status（）；
描　　述：函数 check_delay_status 检查 delay_time 是否正在执行，即批处理过程中正处于延时状态。
返 回 值：如果当前正处于延时状态函数返回 1，否则返回 0。
系　　统：WINDOWS
参　　见：

函 数 名：set_port

目　　的：用于控制输出状态。
语　　法：int set_port（int ch，int state）；
　　ch：所设置输出口号，1 对应于 P3，2 对应于 D3；
　　state：输出状态，1：ON，0：OFF；
调用例子：set_port（1，1）；/*输出口 1 输出 1。*/
描　　述：在缺省情况下，输出口的状态为 0，及 P3+和 D3+为低电平，P3-和 D3-为高电平。当调用 set_port 将输出口的状态设置为 1 时，P3 和 D3 的电平状态随之发生变化。
返 回 值：如果设置成功，则 set_port 返回值为 0，否则返回-1。
系　　统：DOS，WINDOWS
参　　见：

函 数 名：check_EncB, check_EncZ
目　　的：用于检查编码器输入端口的状态。
语　　法：int check_EncB（int ch）；
　　int check_EncZ（int ch）；
　　ch：所检查端口的轴号；
调用例子：check_EncB（1）；/*检查第 1 轴 E1B 的状态。*/
描　　述：当编码器输入口用作 I/O 口时，可以用这两个函数检查输入状态。当函数返回 1 时，相应的输入口为 ON；返回 0 时，相应的输入口为 OFF。
返 回 值：如果设置成功，则 set_port 返回值为 0 或 1，否则返回-1。
系　　统：DOS，WINDOWS
参　　见：

函 数 名：change_speed
目　　的：用 change_speed 函数来实现运动中变速的功能。
语　　法：int change_speed（int ch,double speed）；
　　ch：轴号；
　　speed：变化到的速度。
描　　述：函数 change_speed 函数来实现运动中变速的功能，变速范围的最大值不能超过 set_maxspeed()所设定的值。当以常速运动指令启动运动后，即可调用该函数在运动过程中实现变速。变速时的加速度由运动指令函数调用前的 set_profile 函数参数决定。
返 回 值：调用正确返回 0，错误返回-1。
系　　统：WINDOWS
参　　见：set_profile

函 数 名：Outport
目　　的：用 Outport 函数来对 PC 机口地址进行写操作。

语　　法：int Outport（int portid,unsigned char byte）;
　　int portid：口地址
　　unsigned char byte：输出的一个字节数据
描　　述：函数 Outport 将一个字节的数据写到 portid 对应的口地址，如计算机并口。该函数与 MPC01 卡操作无关，通过该函数可方便地在 VB 中对 PC 机口进行读写操作，VC 中有相关的口操作函数。
返 回 值：函数正确执行返回 0，否则返回-1。
系　　统：WINDOWS
参　　见：

函 数 名：Inport
目　　的：用 Inport 函数来对口地址进行读操作。
语　　法：int Inport（int portid）;
　　int portid：口地址
描　　述：函数 Inport 读地址 portid 对应的输入口数据，如计算机并口。该函数与 MPC01 卡操作无关，通过该函数可方便地在 VB 中对 PC 机口进行读写操作，VC 中有相关的口操作函数。
返 回 值：返回读取的内容。
系　　统：WINDOWS
参　　见：

6.5 常见问题及解决方法

6.5.1 软件中对板卡进行初始化设置方法

（一） 显式设置

在程序初始化部分按如下方式调用 DLL 函数进行设置：

```
……
set_board_num(1);            //设置控制卡数目为 1
set_axe(1,3);                //设置第 1 块控制卡上有 3 个轴
set_board_base(1,0x300);     //设置第 1 块控制卡的基地址为 0x300
set_board_irq(1,3);          //设置第 1 块控制卡中断号为 3（若有多块卡，应分别设置）
init_board();                //按前面设置的参数初始化控制卡
……
```

（二） 自动检测并对板卡进行设置

在程序初始化部分按如下方式调用 DLL 函数进行设置：

```
……
```

```
auto_set();                 //自动设置控制卡的数目、轴数、基地址
set_board_irq(1,3);         //设置第 1 块控制卡中断号为 3（若有多块卡，应分别设置）
init_board();               //按前面设置的参数初始化控制卡
……
```

注意：以上两种方法只需采用其中一种。

以上初始化完成后，各轴速度及模式设置缺省值如下：

参数类型	参数	缺省值	修改函数
常速运动参数	各轴常速度	2000	set_conspeed
	矢量常速度	2000	set_vector_conspeed
快速运动参数	低速	2000	set_profile
	高速	8000	
	加/减速度	80000	
	矢量低速	2000	set_vector_profile
	矢量高速	8000	
	矢量加/减速度	80000	
	升降速类型	梯形	set_s_curve
模式及状态	脉冲输出方式	脉冲/方向方式	set_outmode
	输出信号逻辑	负逻辑	set_outmode
	回原点运动时检测原点信号方式	仅检测原点接近开关信号	set_home_mode
	指令执行方式	立即方式	set_batch_flag
编码器反馈模式	反馈信号模式	A/B 相 90 度相位差	set_encoder_mode
	倍频数	4	
	计数单位	每 1 个脉冲计数 1 次	

之后可根据需要进行模式切换，这些函数需要根据硬件实际情况进行调用设置，缺省设置及函数各参数具体含义见函数描述一章。若使用缺省设置，则程序中可以不调用这些函数。

```
set_batch_flag(1);          //转换到批处理方式
set_outmode(1,1,0);         //脉冲输出模式及信号逻辑设置
set_home_mode(1,0);         //回原点时检测原点信号的方式
```

6.5.2 读取编码器反馈

```
long Enc;
…
set_encoder_mode(1,1,2,1);  //设置 1 轴的编码器反馈信号模式和计数单位，缺省情况下，get_encoder 函数返回的是输出脉冲数；调用该函数之后，get_encoder 函数的返回值将是按照设置的反馈信号模式和计数单位读取的编码器的反馈值。
```

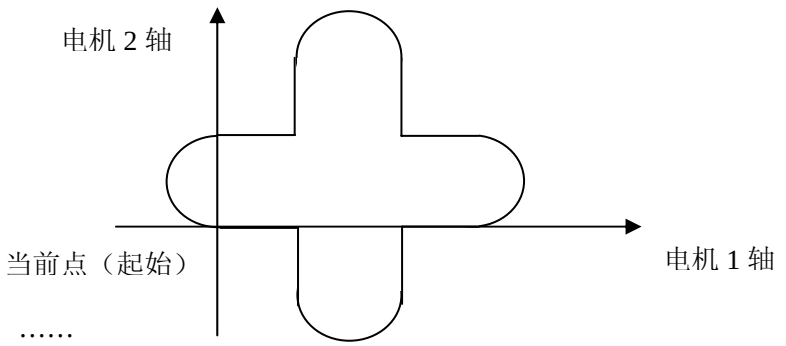
get_encoder(1,&Enc);　　//取 1 轴的编码器反馈，可根据需要在定时器中调用或在循环中调用。

…

关于编码器反馈接线请参考 MPC01 接口一章的内容。

6.5.3 多指令连续运动

下面一段代码演示如何使两轴在二维平面通过多指令连续运动走出如下图形。




```
……
set_batch_flag（1）;　　//切换到批处理模式
set_conspeed（1, 1000）;　　//设置 1 轴常速度
set_conspeed（2, 1000）;　　//设置 2 轴常速度
set_vector_conspeed（1000）; //设置矢量常速度，用于插补函数

arc_center（1, 2, 0, 2000, 180）;
con_pmove（1, 4000）;
con_pmove（2, 4000）;
arc_center（1, 2, 2000, 0, 180）;
con_pmove（2, -4000）;
con_pmove（1, 4000）;
arc_center（1, 2, 0, -2000, 180）;
con_pmove（1, -4000）;
con_pmove（2, -4000）;
arc_center（1, 2, -2000, 0, 180）;
con_pmove（2, 4000）;
con_pmove（1, -4000）;
……
```

6.5.4 运动变速

用 change_speed 函数可以很容易实现运动中变速，以下代码演示实现运动中变速的问题。

定义全局变量

```
double CurSpeed=0;
double MaxSpeed=100000;
```

在“启动”按钮的响应函数中增加如下代码：

```
set_maxspeed(1,MaxSpeed);
set_vector_conspeed(100,1000,1000);
fast_pmove(1,1000000);
```

在“升速”按钮的响应函数中增加如下代码：

```
CurSpeed=CurSpeed+1000;
If(CurSpeed>MaxSpeed) CurSpeed=MaxSpeed;  //限定变速范围不超过最大值
change_speed(1,CurSpeed);
```

在“降速”按钮的响应函数中增加如下代码：

```
CurSpeed=CurSpeed-1000;
change_speed(1,CurSpeed);
```

先单击“启动”按钮启动运动，之后每单击一次“升速”按钮，当前速度增加 1000；每单击一次“降速”按钮，当前速度减小 1000。

也可以用另外一种方式实现变速，即在批处理方式下将升速和变速过程用微线段运动来逼近，通过设置各微线段的不同速度来实现，如下代码演示如何实现简单的变速：

```
double Speed;
double Dis;
int i;
set_batch_flag(1);
for ( i=1 ; i <= 20 ; i ++ )                    //升速部分
{
    Dis = 10000 + i * 1000 ;
    Speed = i * 10000 ;
    set_conspeed ( 1, Speed ) ;
    con_pmove ( 1, Dis ) ;
}
con_pmove 1, 50000                              //匀速部分
for ( i=20 ; i >=1 ; i-- )                      //降速部分
{
    Dis = 10000 + i * 1000 ;
    Speed = i * 10000 ;
    set_conspeed ( 1, Speed ) ;
    con_pmove ( 1, Dis ) ;
}
```

6.5.5 正确判断前一个运动指令是否执行完毕

问题：如下一段代码

```
……
con_pmove(1,1000);              //第一条运动指令：1 轴发出 1000 个脉冲
while(check_done(1) = = -1)     //循环判断 1 轴是否运动完毕，运动结束后执行
                                下一条运动指令
```

con_pmove(1,2000)　　//第二条运动指令：1 轴发出 2000 个脉冲

……

在这段代码中，本希望通过第二行语句来判断第一个运动指令是否执行完毕，直到执行完毕才发出第二条运动指令，但是该段代码将无法得到正确的执行结果，出现只有第一条指令被执行的现象。

原因：

函数库内部采用指令缓冲方式，将应用程序通过 DLL 库函数发出的运动指令压入内部建立的指令缓冲区后便立即返回，因此应用程序调用这些运动指令库函数将很快返回，此时可能运动指令尚未开始执行，因此上述代码中第二行循环判断语句中 check_done()函数返回值为 0 而使得循环退出，随后第 3 行语句发出下一条运动指令，此时该运动指令控制的轴因为前一条运动指令已经启动并且还在运动而使得指令失效，出现只有第一条指令被执行的现象。**注意：该问题只出现在立即方式下，批处理方式下无须用循环判断前一条指令是否执行完毕，即代码中不需要 while 循环。程序会自动在前一条执行完后再执行下一条。当然也可以用这种方式执行，以便在一批指令中间插入一个其它的操作。**

解决方法：可以将上面这段代码做如下处理

……

con_pmove(1,1000);　　//第一条运动指令：1 轴发出 1000 个脉冲

while (check_done(1) = = 0) ;//通过该循环判断第一条运动指令控制的轴是否开始运动

while (check_done(1) = = -1) ;//循环判断 1 轴是否运动完毕，运动结束后执行下一条运动指令

con_pmove (1,2000) ;　　//第二条运动指令：1 轴发出 2000 个脉冲

……

这段代码中第一重循环主要目的是确保第一条运动指令控制的轴已经开始运动，以便随后的循环判断运动的轴是否运动完毕。

这种方法实际应用时应根据情况对两重循环特别是第二重循环的代码作相应处理，以避免因为长时间循环等待而无法对 Windows 其它消息作出响应，出现类似死机的现象。在 VB 中只需简单地在循环体中添加如下语句：

DoEvents

这种方法虽然简单，但也有不足之处，主要是：

1）对于某些指令脉冲数很少，如以高速运动几个脉冲距离的运动指令，第一重循环可能无法退出而陷入死循环。

2）对于圆弧插补，第二重循环可能无法判断到运动结束而提前退出。

3）无法处理这种情况，即在一批指令按照批处理执行方式发出后要循环等待所有已发出指令执行完毕，之后再进行其它操作。

以下代码可以很好地解决这些问题。

```
long Pos1,Pos2;
…
set_batch_flag(1);  //切换到批处理方式，以便 check_batch_done()能够正确判断
get_abs_pos(1,&Pos1);
con_pmove(1,1000);//发出第一条指令,也可以发出多条指令按照批处理方式执行
do{
      get_abs_pos(1,&Pos2);  //位置发生了改变
}while(Pos1= =Pos2);
while ( check_batch_done ()!=0 ) ; //直到前面发出的所有指令完成后终止循环
con_pmove(1,1000);              //发出下一条指令
```

6.5.6 减速、原点信号的使用

在某个轴的梯形速度（由 set_profile 设置）运动过程中，如碰到减速开关，则该轴的运动速度将自动从高速减到低速，并保持低速运行。一般情况下，减速开关与原点接近开关配合使用，以提高在高速回原点时的定位精度。将减速开关安装在原点接近开关的前面，在高速回原点时，运动机构先碰到减速开关，使之减速，并以低速靠近原点接近开关，在这个过程中，减速开关信号应保持闭合有效，在到达原点位置时，停止运动。如果没有减速开关，在高速回原点时，碰到原点接近开关将立即停车，由于运动机构的惯性、原点接近开关的有效工作范围等因素，将会降低回原点的定位精度，并可能给机械造成冲击。

注意：这里的正、负向是指控制卡发送脉冲的方向，可能与运动机构的实际运动方向并不一致。

6.5.7 如何提高速度精度

在 MPC01 卡的使用中，有时发现在运动时用 get_rate 读取的频率与设置的脉冲频率差别较大，其原因如下：

MPC01 卡的输出脉冲频率由两个变量控制：脉冲分辨率和倍率，两者的乘积即输出的脉冲频率。由于倍率寄存器长度是有限的，即 13 位（最大值为 8191），如果要达到 2400KHz 的最大输出脉冲频率，脉冲分辨率应为（2400000/8191）=293Hz，如果实际使用的脉冲频率为 100Hz，显然 MPC01 卡只能输出一个分辨率的脉冲频率（即 293Hz）。为了解决这个问题，可以调用 set_maxspeed 设置需要达到的最大输出脉冲频率。比如：set_maxspeed（1，1000），设置后脉冲分辨率将被重新设置，为（1000/8191）=0.12Hz，这样就能满足低速时速度精度的问题。注意：MPC01 卡的最高分辨率可以达到 0.01Hz，但此时 MPC01 卡的最大输出频率只能达到 81.91Hz。

6.5.8 如何实现方向信号超前于脉冲信号

某种品牌的步进电机驱动器在控制时序上要求方向信号要比脉冲信号超前 500 微秒，用 MPC01 卡控制时出现这种现象：当发出反转指令时，电机会向原来的方向转动一点，然后才反转，造成位置不准。为什么会出现这种现象，怎么处理？

有些步进电机的驱动器在控制时序上要求方向信号要比脉冲信号超前一定时间（几十到几百微秒），否则将工作不正常。而 MPC01 卡的脉冲信号和方向信号基本上是同时发出的，所以当发出反转指令时，驱动器的方向信号还没完全翻转时就接

收到了脉冲信号，故而电机会向原来的方向转动一点，在驱动器的方向信号完全翻转后，电机才反转。对于这种驱动器，在调用运动指令前，先设置将要运转的方向：set_dir（ch, dir），延时足够的时间，确保驱动器的方向信号稳定后，再调用运动指令。注意：在运动过程中不要调用 set_dir，否则会导致电机突然反转。

6.5.9 如何利用 Z 脉冲实现精确回原点

A.接线：将原点开关信号接在 ORGx 端；将 Z 脉冲信号接在 Ex Z 端；其中 x 为轴号。B.调用指令：在 init_board（）之后调用 set_home_mode（x, 1）；将 x 轴的回原点方式设置为原点信号和 Z 脉冲信号同时有效的方式。然后调用 con_hmove 或 fast_hmove 回原点指令，即可实现精确回原点。

6.5.10运行 EXE 文件时系统显示找不到 DLL 文件

可能是用户尚未正确安装 MPC01 软件，请按照 6.3.2 节内容安装软件。

注意：安装程序将把库文件 Dmc.dll 和 Dmc.vxd 安装到 Windows 系统目录下（若 Windows 安装在 C:盘 Windows 目录下，则系统目录为 C:\windows\system），其余文件安装到在安装过程中指定的安装目录下。这样便可以在任何目录下使用函数库。但在升级函数库时应注意，参见 6.5.6 节。

6.5.11如何将开发的软件系统制作成安装程序后发行给最终用户

在制作安装程序时，请将如下文件打包进安装程序：

（1） Dmc 动态链接库文件 dmc.dll

（2） 虚拟设备驱动程序 dmc.vxd

（3） VC 动态链接库文件 msvcrt.dll 和 mfc60.dll

6.5.12运行 EXE 文件后死机

可能是以下原因造成：

（1） 板卡中断设置与其他设备中断设置有冲突，此种情况表现为 Windows 启动后即死机或发生异常，如鼠标不动等。此时请关机后取出板卡重新进行中断设置，空闲中断号及 IO 口地址可根据 Windows 系统信息获得。

（2） 板卡中断号及基地址设置与程序中软件设置不一致，此种情况表现为 Windows 启动正常，但一运行与 MPC01 相关的程序即死机。此时请仔细核对板卡设置与软件设置。

6.5.13软件能够正常启动，但无法产生运动

请仔细按以下几个方面检查：

（1） 电机、驱动器等执行机构是否完好；

（2） 执行机构与计算机是否已经正确连接连接；

（3） 板卡是否已经插入计算机并正确设置了基地址及中断号；

（4） 软件中是否调用了初始化函数进行了初始化；
（5） 软件中是否调用了中断号设置函数进行中断设置，所设置中断号是否与板卡上跳线一致。
（6） 软件中运动指令参数是否正确；
（7） 调用错误代码获取函数获取错误代码后进行分析；

6.5.14 如何在 Windows 9X 操作系统环境下正确设置 MPC01 卡中断号

由于 Windws 9X 操作系统支持即插即用，因此对 PC 机中支持该特性的插卡会自动进行中断号等资源设置。MPC01 卡暂不支持该特性，需要在板卡上进行跳线设置中断号，因而如何获知系统已经分配的中断号，避免使 MPC01 卡设置的中断号与系统其它设备的中断号重复，便是一个必须注意的问题，可按如下几种方法进行：

- 打开 Windows 控制面板，双击“系统”图标，在“系统属性”对话框的“设备管理器”选项卡设备列表中选中某设备，单击“属性”按钮可查看该设备所占用的中断号等资源。
- 在“开始”——“程序”——“附件”——“系统工具”菜单栏下有“系统信息”菜单项，单击可打开“Microsoft 系统信息”窗口，在窗口中可查看硬件资源，如 IRQs 及 I/O 等，从而找出尚未占用的中断。若菜单中没有“系统信息”一项，则可能是安装 Windows 时未安装该部分，可通过 Windows 控制面板中“添加/删除程序”将该部分安装进来。

注意：在进行以上操作前请暂时不要插 MPC01 卡或设置卡中断号，否则可能导致与系统其它硬件设备中断号冲突而无法正常启动 Windows。

PC 机常用口地址及中断号资源可参见 8.4 和 8.5 节内容。

在找不到其它空闲中断号时，可以用串口 COM1（中断号为 4）或 COM2（中断号为 3）的中断号，但应注意，若串口上插有其它设备（如串口鼠标或 Modem）时则不能用插有设备的串口的中断号，否则该设备将无法正常工作）。

6.5.15 如何升级函数库

请您经常访问本公司的网站（http://www.step-servo.com）以下载获取最新版本的驱动程序及函数库，新版本函数库将会保持与旧版函数库已有函数的兼容，并根据需要增加新的函数。

若您获得一套最新的函数库，您可以按照以下方法对您的旧函数库进行升级：

（1） 检查您获得的函数库是否完整。完整的函数库应包含 Dmc.dll 和 Dmc.vxd 两个文件；
（2） 关闭与 MPC01 相关的正在运行的所有程序；
（3） 将这两个文件拷贝覆盖到 Windows 系统目录下，如：假设 Windows 安装在 C:盘 Windows 目录下，则将这两个文件拷贝到 C:\windows\system 目录下；

注意： 也可以将这两个文件拷贝到与您的 EXE 文件相同的目录下，但此时应确保Windows系统目录下没有这两个文件，否则计算机将先搜索Windows系统目录，然后才搜索当前目录，从而使得 Windows 系统目录下的旧版函数库被先加载而无法正常运行。

6.6 错误代码表

错误代码	代码含义	备注
0	正确	
1	系统缺省错误	
2	调用 GetErrorCode()函数时所给参数超出了 1 到 10 的范围	
1001	VXD 缺省错误	
1002	当前指令操作的轴正在进给	
1003	Vxd 未正确初始化	
1004	自动检测出的板卡数为 0	
1005	Vxd 加载错误	
2001	初始化变量时内存分配错误	
2002	板卡数设置错误	
2003	板卡基地址设置错误	
2004	板卡轴数设置错误	
2005	板卡中断号设置错误	
3001	设置输出模式参数错误	
3002	设置回原点模式参数错误	
3003	设置常速度参数错误	
3004	设置梯形速度参数错误	
3005	设置 S 型曲线参数错误	
3006	当前卡为 B 型卡无法设置 S 曲线	
3007	设置最大速度参数错误	
3008	set_get_pos()参数错误	
3009	set_encoder_mode()参数错误	
3010	连续运动指令参数错误	
3011	点位运动指令参数错误	

6.7 多指令连续运动时的升降速处理

6.7.1 功能说明

在多条运动指令连续执行时，可进行特殊升降速处理，即第一条运动指令只有升速过程而无降速过程，中间运动指令既无升速过程也无降速过程，最后一条指令只有降速过程而无升速过程。

6.7.2 相应库函数描述

函 数 名：**set_ramp_flag**
目　　的：用于在多条运动指令连续执行时进行特殊升降速处理的设置。
语　　法：int set_ramp_flag (int flag);
调用例子：err= set_ramp_flag (1);　　/*设置下一条指令为特殊处理的第一条指令*/
　　err= set_ramp_flag (2);　　/*设置下一条指令为特殊处理的最后一条指令*/
返 回 值：如果设置成功，则返回值为 0，否则返回-1。
系　　统：WINDOWS
参　　见：

6.7.3 使用方法及应注意的问题

6.7.3.1 使用方法

对于要进行特殊升降速处理的一批指令，在第一条运动指令前调用库函数，函数参数为 1，如：set_ramp_flag(1)；在最后一条运动指令前调用库函数，函数参数为 2，如：set_ramp_flag(2)即可。

如下一段代码为一段示例代码：

```
int i;
……
set_ramp_flag(1);                  //设置特殊升降速处理起始标记
fast_line2(1,20000,2,20000);
fast_arc_center(1,2,10000,-10000,180); //对两段直线运动进行圆弧过度平滑处理
                                       //否则可能因拐点处矢量速度发生突变产
                                       //生冲击
fast_line2(1,-20000,2,-20000);
set_ramp_flag(2);                  //设置特殊升降速处理结束标记
```

fast_arc_center(1,2,-10000,10000,180);

……

6.7.3.2 注意的问题

（1） 该方法只对如下指令有效：

- fast_line2()
- fast_arc_center()

若要实现 fast_pmove(1,10000)，可以用如下指令代替：

fast_line2(1,10000,2,0)

（2） 请严格按照使用方法中代码顺序及位置调用 set_ramp_flag()库函数，并传递正确的参数，否则将可能出现不正确的升降速。

（3） 在该段代码中间不要调用 set_batch_flag()库函数进行立即方式和批处理方式之间的切换，系统在该段代码执行期间将自动设置为批处理方式。若要切换，请在该段代码之前或之后进行。

（4） 该功能以 3.2 版函数库为基础，因此必须要进行中断的设置，详细使用方法请参阅 2.1.2 节。

（5） 当两直线连接处形成拐点时，建议用圆弧进行平滑过度处理，否则将可能因拐点处矢量速度发生突变而产生冲击。

7 函数索引

arc_certer ---------- 34
arc_final ---------- 35
auto_set ---------- 25
change_speed ---------- 54
check_alarm ---------- 53
check_batch_done ---------- 51
check_board ---------- 39
check_delay_status ---------- 53
check_done ---------- 39
check_EncB ---------- 54
check_EncZ ---------- 54
check_limit ---------- 39
check_SD ---------- 53
check_status ---------- 38
con_hmove ---------- 31
con_hmove2 ---------- 31
con_hmove3 ---------- 31
con_line2 ---------- 33
con_line3 ---------- 33
con_pmove ---------- 30
con_pmove2 ---------- 30
con_pmove3 ---------- 30
con_vmove ---------- 30
con_vmove2 ---------- 30
con_vmove3 ---------- 30
decel_stop ---------- 36
decel_stop2 ---------- 36
decel_stop3 ---------- 36
delay_time ---------- 53
disable_axis_intr ---------- 41
disable_board_irq ---------- 40
disable_timer_irq ---------- 42
enable_axis_intr ---------- 41
enable_board_irq ---------- 40
enable_timer_irq ---------- 42
end_backlash ---------- 43
fast_arc_center ---------- 34
fast_hmove ---------- 31
fast_hmove2 ---------- 31
fast_hmove3 ---------- 31
fast_line2 ---------- 33
fast_line3 ---------- 33
fast_pmove ---------- 30
fast_pmove2 ---------- 30

fast_pmove3 ---------- 30
fast_vmove ---------- 30
fast_vmove2 ---------- 30
fast_vmove3 ---------- 30
get_abs_pos ---------- 37
get_board_base ---------- 24
get_board_irq ---------- 26
get_conspeed ---------- 27
get_cur_dir ---------- 52
get_encoder ---------- 37
get_err ---------- 51
get_last_err ---------- 51
get_max_axe ---------- 24
get_profile ---------- 28
get_rate ---------- 28
get_rel_pos ---------- 37
get_vector_conspeed ---------- 32
get_vector_profile ---------- 32
helical_move ---------- 35
init_board ---------- 25
Inport ---------- 55
Outport ---------- 54
reset_err ---------- 51
reset_pos ---------- 43
set_abs_pos ---------- 52
set_axe ---------- 24
set_backlash ---------- 43
set_batch_flag ---------- 50
set_board_base ---------- 24
set_ board_irq ---------- 26
set_ board_isr ---------- 41
set_board_num ---------- 23
set_conspeed ---------- 27
set_dir ---------- 52
set_ellipse_ratio ---------- 33
set_encoder_mode ---------- 37
set_getpos_mode ---------- 37
set_home_mode ---------- 27
set_maxspeed ---------- 27
set_outmode ---------- 26
set_port ---------- 53
set_profile ---------- 28
set_s_curve ---------- 29
set_s_section ---------- 29
set_ timer_isr ---------- 42
set_vector_conspeed ---------- 32
set_vector_profile ---------- 32
start_backlash ---------- 43
sudden_stop ---------- 36

sudden_stop2 ---------- 36
sudden_stop3 ---------- 36
wait_done ---------- 43
wait_done2 ---------- 43
wait_done3 ---------- 43

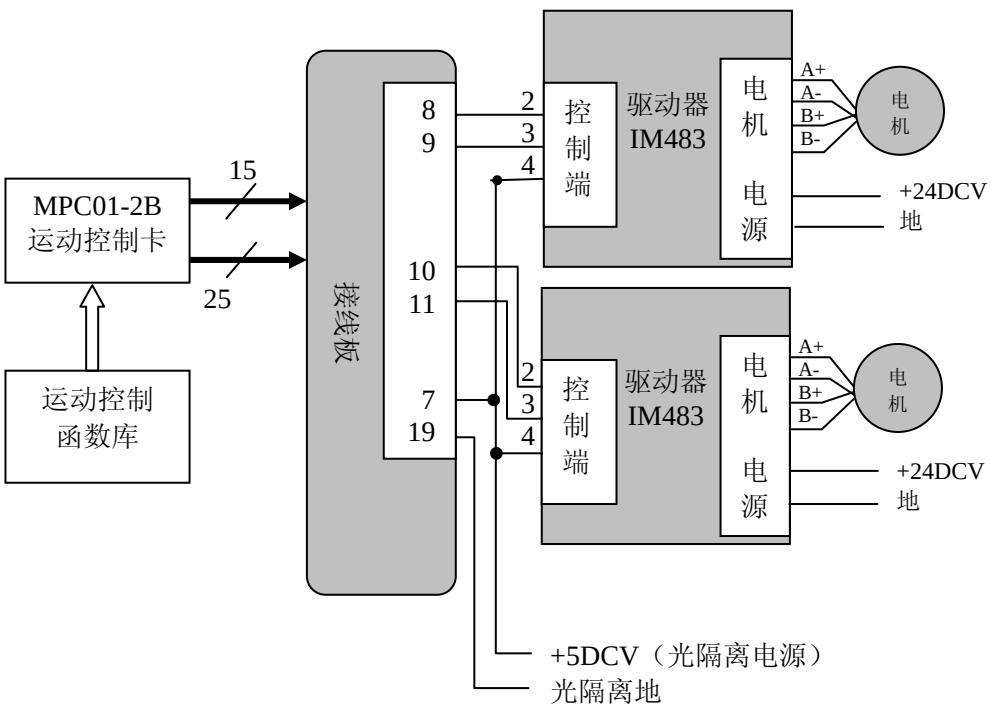
8 附录

8.1 两轴步进控制系统示例

8.1.1 系统配置

1. 混合式步进电机：意大利 HN200 3426（1.8°，2.8N.m）；
2. 驱动器：美国 IMS 公司 IM483（最大细分 256，峰值电流 4A）；
3. 上位控制：MPC01-2B；
4. 直流开关电源：24DCV（10A），5DCV（1A）。

8.1.2 控制电路接线图



*关于步进电机 HN200 3426 和驱动器 IM483 的应用请参考相应的使用说明书。

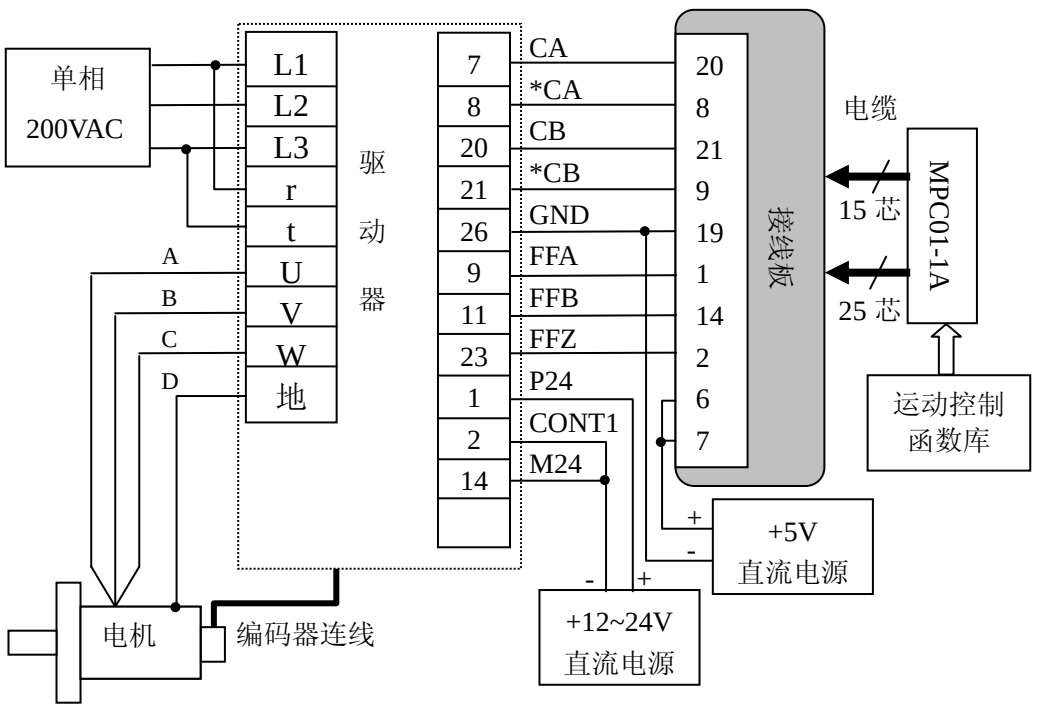
8.2 单轴数字式伺服控制系统示例

全数字式交流伺服系统（例如：松下交流伺服系统 MINAS A 系列或富士 FALDIC-β 系列）可以接收脉冲和方向信号来控制其运动过程，因此，可以用 MPC01 卡来实现对伺服电机系统的控制。下面以单轴伺服控制系统为例简要说明 MPC01 与伺服系统之间的连接方法。

8.2.1 系统配置

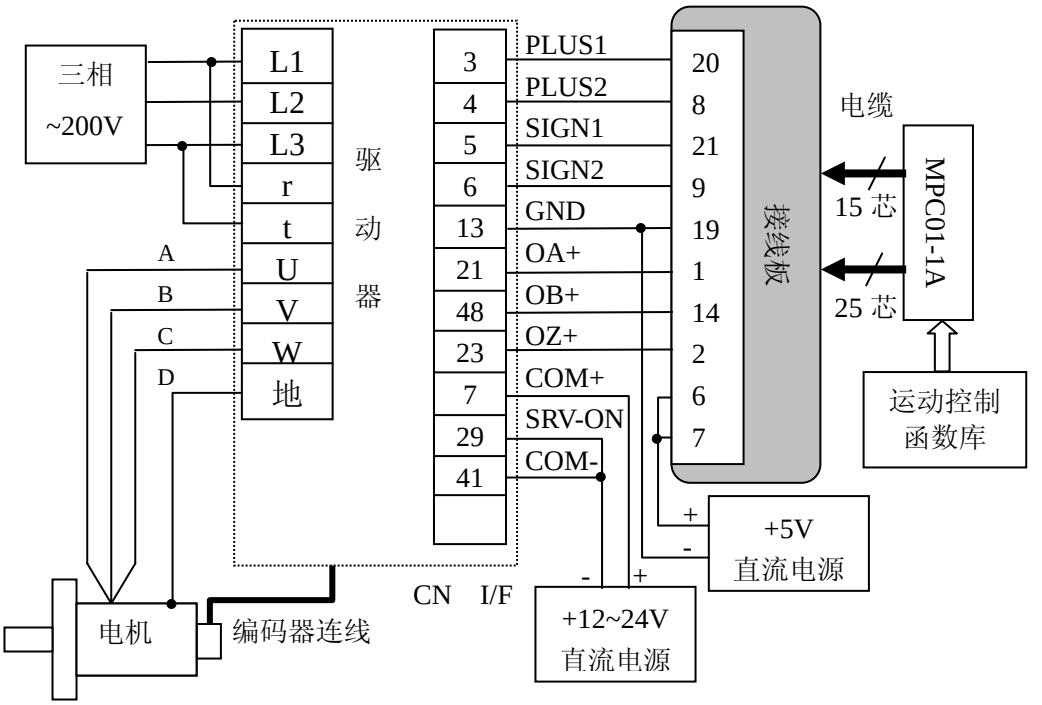
1. 伺服系统：富士 FALDIC-β 系列或松下 MINAS A 系列全数字式交流伺服系统任意型号；
2. 控制卡：MPC01-1A；
3. 直流开关电源：24DCV（1A），5DCV（1A）。

8.2.2 控制电路接线图



MPC01-1A 与 FALDIC-β 系列的接线图

*有关富士全数字交流伺服系统的应用请参考其说明书。



MPC01-1A 与 MINAS A 系列的接线图

*有关松下全数字交流伺服系统的应用请参考其说明书。

8.3 PC 打印机口用作 I/O 口

PC 打印机接口为并行 I/O 口，与其它通用 I/O 口性质完全一样。它由一个 25 芯的 D 型接口提供 TTL 输入和输出信号，其引脚定义如下：

引脚	I/O 方向	打印机中的功能	信号说明
1	Out	-STROBE	数据输出触发
2~9	Out	Data bits（D0~D7）	输出数据到打印机
10	In	-ACK	应答
11	In	BUSY	忙
12	In	PE	纸尽
13	In	SLCT	打印机选择
14	Out	-Auto FD XT	自动回车换行
15	In	-ERROR	出错
16	Out	-INIT	初始化打印机
17	Out	-SLCTIN	选择数据输出到打印机
18~25	-	Commond ground	公共地

在上表中的输入输出信号共占用 PC 机的三个 I/O 地址，对于 LPT1（并行打印机口 1）而言，这三个 I/O 地址的各位（bit）定义如下：

I/O 地址	**信号种类**	**位**	**功　　能**	**DB25 引脚**
378H（输出）	数据信号	D0~D7	输出至打印机	Pin2~Pin9
379H（输入））	状态信号	D0~D2 D3 D4 D5 D6 D7	没使用 -ERROR SLCT PE -ACK BUSY	 Pin15 Pin13 Pin12 Pin10 Pin11
37AH（输出）	控制信号	D0 D1 D2 D3 D4 D5~D7	-STROBE -Auto FD XT -INIT -SLCTIN 中断允许（IRQ7） 没使用	Pin1 Pin14 Pin16 Pin17

由上表可见，一个打印机口，总共有 12 根输出和 5 根输入可供使用，一般能够满足需要少量 I/O 信号的场合。注意：379H 口的第 7 位（BUSY）在打印机接口电路中是从连接器经反相后接到总线上的；同样，37AH 口的 0、1、3 位也是经反相后接入总线的，在使用中应注意区分。

8.4 PC 机 I/O 地址分配

PC 机系统支持的端口地址范围是从 0~3FF，共 1024 个端口地址，有效译码地址信号是 A9~A0。其中前 512 个端口（0~1FFH）被系统板所占用，因此其它扩展板卡一般不应该使用这些端口；高端的 512 个端口（200~3FF）是为了扩展板卡预留的，但有一些已被标准扩展板所占用，例如：单显适配器占用 3B0H~3BFH，彩色图形适配器占用 3D0H~3DFH，因为这些扩展板卡是 PC 机的基本配置，所以这些端口用户不能使用。下表是 PC 机 I/O 端口地址的分配表，在表上，地址 200H 以后注明保留的或未出现在该表中的端口地址可以由用户使用。

PC 机 I/O 端口分配表

分类	地址范围（16 进制）	I/O 设备端口
系统板	000~01F	DMA 控制器 1
	020~03F	中断控制器
	040~05F	定时器/计数器
	060~06F	键盘
	070~07F	实时时钟
	080~09F	DMA 页面寄存器
	0A0~0BF	中断控制器 2
	0C0~0DF	DMA 控制器
	0F0~0FF	协处理器
	100~1EF	未用
	1F0~1F8	硬盘
I/O 通道	200~20F	游戏接口
	258~25F	Intel Above Board
	278~27F	并行打印口 2
	280~2DF	Ultimate EGA
	2E1	GPIB
	2E2~2E3	Data Acquisition
	2F8~2FF	串口 2
	300~31F	试验板
	360~36F	PC Network
	378~37F	并行打印口 1
	380~38F	SDLC 通信
	390~393	Cluster
	3A0~3AF	Binary Synchronous Communication
	3B0~3BF	单显适配器
	3C0~3CF	EGA
	3D0~3DF	彩色图形显示适配器
	3F0~3F7	软盘驱动器
	3F8~3FF	串口 1

8.5 PC 机中断线分配

每台 PC 机总共有 15 根有效的中断线，其中许多已被占用，下表列出了 PC 机中断线的一般分配情况，在使用时注意选用。一般来说，在工控系统中，有些外围设备并不需要，如网卡、打印机等，所以中断线 IRQ2、3、5、7、10、11、12、15 都能由用户使用，但为了保险起见，在使用某个中断线之前，最好核实它是否已被占用，以免发生冲突。

PC 机中断线分配表

IRQ 号	**用　　途**
IRQ2	等于 IRQ9，通常可用
	EGA Display Adapter
	PC Network
IRQ3	COM2
	PC Network
	Binary Synchronous Communication
	Cluster
	通常可用
IRQ4	COM1
	Binary Synchronous Communication
	SDLC
IRQ5	通常可用
IRQ6	Floppy Disk
IRQ7	打印口 LPT1
	Cluster
	通常可用
IRQ8	DOS 定时器
IRQ10	Open
IRQ11	Open
IRQ12	Open
IRQ14	Fixed Disk
IRQ15	Open